On The Relationship between Knowledge-based Systems Theory and Application Programs: Leveraging Task Specific Approaches[*]

Jon Sticklen and Eugene Wallingford

AI/KBS Laboratory • CPS Department • Michigan State University
East Lansing, Michigan 48824-1027 USA

Abstract. The relationship between theories of knowledge-based problem solving and application-level programs is not well understood. The traditional view has been that given some knowledge-based systems theory, a successful application program built following the theory provides strong support for the theory. This viewpoint fails largely because the link between theory and application is totally through the human implementer of the application program. Insight for how to cope with this problem can be obtained from the Knowledge Level Hypothesis (KLH) of Newell. But in order for the KLH to be helpful, we must extend it to incorporate concepts of control knowledge. After describing the theory/application linkage problem, we go on to give an overview of the task specific approaches to knowledge-based system. We then discuss both Newell's KLH and an extension to it that will help in solving the AI Theory/AI application linkage problem. We end with the recommendation that knowledge-based systems theory could be grouped with those disciplines in which theory verification by experimental inquiry is the norm.

0.1. Introduction

The relationship between knowledge-based systems theory and application-level programs is not well understood. In the early "roaring 80's," rule-based systems were thought to be capable of expert level problem solving in any selected domain, and the link between theories of expert-level problem solving and applications was perceived to be 1:1. That is, when a researcher developed a theory of problem solving P, he "tested" P by implementing a problem solver performing some task P_1 exercising P. In many cases,

[*] Sticklen and Wallingford gratefully acknowledge the support of DAPRA (ARPA 8673), the NSF Center for High Speed Low Cost Polymer Composites Processing at MSU (EEC-9108846), the McDonnell Douglas Research Laboratories, and generous equipment support from Apple Computer.

starting with the MYCIN applications themselves [1, 2], the implemented systems were built following a rule-based approach. The difficulty was that the theory under test when using a rule-based approach is fundamentally that "expert-level, domain-oriented problem solving can be captured in a rule-based approach."

That claim – that any problem solving can be captured in a rule based approach – is not illuminating because it only paraphrases the Turing Thesis, provided we assume that expert level problem solving is a computable function. The Turing Thesis is that an abstract computational engine (the Turing Machine) can compute anything that is computable. The Turing Machine is a bridge between mathematical theory and the realization of computational machines. Although embodying a different architecture from Turing Machines, Post Production Machines were shown to be equivalent in computational power to Turing Machines. Finally, Post Production Machines are an abstract computational model which can be realized in rule-based system shells. Given the Turing Thesis, a rule-based system can in fact compute *anything* that is computable. Thus, linking a particular AI theory to the performance of a rule-based implemented program has no justification.

In the early 80's many conference and journal papers were devoted to describing working knowledge-based systems (see AAAI-80 for example). By the end of the decade, a much smaller percentage of conference and journal papers appeared which described implemented systems (see AAAI-90 for example). In fact, in some AI quarters it is now very unfashionable to talk about implemented systems.

In this position paper, we argue that we currently risk "throwing out the baby with the bathwater" – that knowledge-based systems is an area in which experimental testing of theory is natural provided we take a high level view of problem solving. We will argue that the broad task-specific architectures (TSA) approach to knowledge-based systems supports such a high level perspective. We claim that an extension to the Knowledge Level Hypothesis of Newell is necessary to provide the link between knowledge-based system theory and program artifact, and we describe one such extension.

0.2. Background

The lineage of our argument lies in three areas: reflection on the first generation of knowledge-based systems, the task specific architecture (TSA) reaction to shortcomings in the first generation, and finally the attempt by Newell to free discussion of problem solving systems from vocabulary of implementation, the Knowledge Level Hypothesis.

0.2.1. Experience from the First Generation

The roots of knowledge-based systems are in problems of *search*. For typical problems, the use of weak problem solving methods gave rise to very high search complexity forcing the application of heuristic knowledge to control search. By encoding such control knowledge within the same “universal architecture” in which the weak methods were built, the goal was to retain many of the advantages of these architectures: modularity, uniform representation, and a single control strategy among them. The use of unitary architectures lead to unexpected disadvantages as well (see, for example, [3]). In particular, there were two difficulties:

- important control issues were hidden behind clever programming artifices at the implementation-language level, and
- system builders encountered the need to organize knowledge in the system using constructs outside the formalism provided by the architecture, such as MYCIN’s *context hierarchy* [1, 2] and PROSPECTOR’s *models* [4].

Even if a knowledge-based system could be built at the level of a unitary architecture, the *conceptual* problems of analysis and design would not vanish; problems at higher levels of abstraction need to be addressed.

Studies in Knowledge Acquisition (KA) yielded important advances in understanding the use of specialized vocabularies for representing knowledge-based problem solving. The knowledge acquisition problem was described in terms of the *representation mismatch* between the conceptual constructs of human experts and the implementation primitives used to analyze and build KBSs, a representational gap that limited the ability of domain experts to play a direct role in the construction and maintenance of systems [5]. This perspective led to languages and architectures that provide conceptual primitives closer to the task-level abstractions of experts “to reduce representation mismatch from the implementation side” [6]. The task-level architectures produced from this perspective provided greater power for knowledge acquisition because they incorporated explicit representations of the *types* of knowledge expected from those specializing in particular tasks. This enabled the construction of user interfaces for knowledge acquisition systems that relied on simpler syntactic techniques which presupposed the existence of the necessary conceptual structures. In the end, the high-level primitives thus employed could be mapped onto the implementation primitives of a unitary architecture without exposing domain experts to the vagaries inherent in programming such architectures.

Two intuitions grew from common experience with first generation approaches: (1) Certain knowledge and control structures may be common to a particular task (say, design or diagnosis) across different domains, and (2) the structures for different task types will likely differ. Both retrospective analysis of existing systems and prospective design of new systems indicated that an effective KBS will contain — either explicitly or implicitly — a model of the problem solving process it realizes. The evolving task-specific approach recognized the advantages of representing *explicitly* the conceptual organization of domain knowledge assembled to solve a particular type of problem following a given method. This approach denoted a paradigm shift away from use-independent, uniform knowledge bases toward a view of KBSs as collections of diverse conceptual structures organized for use in targeted ways. Weak methods are appropriate when no further knowledge of a domain is available, but typically expertise in a domain affords a more meaningful understanding of how knowledge is used to solve problems efficiently. This new outlook signified one of the central lesson of the first generation.

0.2.2. TSA Viewpoints

Through the 1980's there were a number of research efforts aimed at solving difficulties met in computation-universal approaches to knowledge-based systems. The *Generic Task* (GT) approach of Chandrasekaran and his colleagues has evolved as one of these efforts. The assumption of the GT approach is that knowledge takes different forms depending on its intended function [7-10]. Following the Generic Task view, a problem is analyzed according to the methods associated with solving it, where each method can be specified by the forms of knowledge and inference necessary to apply the method, and by the subproblems that must be solved to carry it out. These sub-problems can then be recursively decomposed in a similar fashion. The assumption of the GT approach is that there exist a number of ubiquitous *combinations* of method, knowledge structure, and inference structure — termed generic tasks — that serve as sub-problems for a variety of complex problem-solving tasks in a variety of domains. The totality of domain knowledge for solving a given problem is viewed as a composition of generic task "agents" that interact based upon their functions and information needs.

Another prominent view of problem solving in knowledge-based systems was due to Clancey. After recognizing that the control strategies implicit in the MYCIN/GUIDON knowledge base could be expressed independent of domain terminology [11], Clancey isolated *heuristic classification* as a method for performing diagnosis and other selection tasks [12-14]. This method decomposes selection tasks into a set of high-level subtasks that characterize the type of problem solving performed by many existing KBSs. By moving to this more abstract level of description, Clancey and his colleagues were able to reformulate MYCIN into NEOMYCIN, a system whose control knowledge made no reference to the application domain and constituted an abstract model of inference independent of implementation.

McDermott and his colleagues have formulated a view of expert problem solving, based on the notion of *role-limiting methods* (RLMs), that is strongly driven by experiences in knowledge acquisition. The RLM approach posits that a large knowledge base can be constructed, maintained, and understood more fruitfully by organizing it according to the various roles that different kinds of knowledge play. On this view, "each role-limiting method defines the roles that the task-specific knowledge it requires must play and the forms in which that knowledge can be represented" [15]. Like Chandrasekaran and Clancey, McDermott holds that families of tasks exist for which the problem solving method and its control knowledge can be abstracted away from the peculiarities of an task instance. This approach, though, focuses its concern with these methods on how they circumscribe the roles and representation of the task-specific domain knowledge on which they operate. The goal of this research program is to identify task families having these characteristics, to abstract their methods, and then to construct an architecture that assists knowledge acquisition for the corresponding tasks. For the purpose of knowledge acquisition, the RLMs represent an important class of methods because they direct the acquisition process at a more abstract level while still providing a broad coverage of tasks in a variety of domains.

Structured methodologies for the construction of application systems following the TSA viewpoint have been developed mainly in Europe. Steels [16] has advocated a framework for system analysis and development with some similarities to Chandrasekaran's task-oriented approach. Following Steels, one first conducts a thorough task analysis in which the task is decomposed into subtasks based on the nature of their inputs and outputs and on the nature of the mappings among them. Second, one constructs a model of the domain knowledge available to perform the task and subtasks. Finally, one applies problem solving methods geared to solving individual subtasks and to structuring subtasks in the pursuance of higher-level tasks. The method selected for each task depends on the kind of knowledge available to solve the task, as captured in the domain model. This methodology differs from that espoused by Chandrasekaran and McDermott, however, in that it allows for a representation of domain knowledge — in the domain model — independent of the method to be selected.

Founded on similar intuitions, KADS is a methodology for the construction of knowledge-based systems that offers an explicit software life cycle and a set of languages for describing and creating KBS structures. This methodology rests on the assumption that task methods share "ways of using knowledge" at a level of abstraction higher than that of concepts in particular domains [17]. The languages in KADS support the development of a *conceptual model* of the problem solving process and a *design model* of the target KBS at a level of abstraction corresponding to the types of knowledge employed. KADS proposes a four-layer representation of knowledge: (1) a domain layer of domain-dependent concepts, relations, and structures; (2) an inference layer that describes what inferences can be made in terms of the *roles* that domain-level entities play; (3) a task layer that controls when inferences are made in terms of goal structures; and (4) a strategy layer for goal generation and task monitoring. Like Steels' approach, KADS allows "task-neutral" representation of domain knowledge but then stresses the importance of having high-level task structures through which to view problem-solving knowledge. These structures include primitive "knowledge sources" at the inference level for solving particular subtasks and goal structures at the task level for representing task decompositions.

0.2.3. The Knowledge Level Hypothesis

In his AAAI presidential address of 1980 [18], Newell proposed a distinct level of analyses for problem solving systems above the symbol level. In the *Knowledge Level Hypothesis*, an implementation-free framework for analyzing problem solving agents was suggested. Since its introduction, the term “Knowledge Level” has become pervasive in the AI literature, particularly in the Expert Systems field. In broad terms, the Knowledge Level Hypothesis has been important in promoting a gradual shift in emphasis away from purely representational issues and toward implementation free descriptions of problem solving.

> ***The Knowledge Level Hypothesis (KLH):*** There exists a distinct computer systems level, lying immediately above the symbol level, which is characterized by knowledge as the medium and with the principle of rationality as the law of behavior.

The most important parts of Newell's hypothesis are as follows:

- The entire information processing system is identified at the Knowledge Level as the “agent.” By identifying the agent as the total system at the Knowledge Level, Newell implicitly acknowledges the view that at the Knowledge Level the separability of problem solving engines and the bodies of domain knowledge they utilize is illusory. Rather, in terms that can be useful at the Knowledge Level, problem solving and domain knowledge are intimately entangled, and the agent must be analyzed on this basis. This opens the issue of the appropriate granularity at which to describe Knowledge Level agents, a discussion to which we will return.
- Knowledge has a generative flavor and cannot be captured in a static data structure; i.e., cannot be expressed at the symbol level. The emphasis that knowledge itself is unbounded and the resultant importance of the generative nature of Knowledge Level constructs is an outgrowth of Newell’s goal of shifting attention away from representation issues: issues that are properly dealt with at the symbol level.
- The attribution of knowledge to an agent is via a process of simulation by self; i.e., a simulation with the “self” being the agent and an assumption by “self” of the goals and symbol-level structures of the other to produce the same actions as the other.
- The central role of self simulation in the attribution of knowledge to an agent is both a source of substantial power in the Knowledge Level Hypothesis, as well as a source of considerable weakness. The power comes *via* the identification of how we may know that an agent has certain knowledge even though we are unable to look inside that agent. The

weakness comes because by using “self” as the simulator, we cannot produce reliable predictions for future behavior.

The incompleteness of the Knowledge Level Hypothesis is shown by its lack of guidance for making predictive statements for some problem solver. Suppose we subscribe to the hypothesis of the Knowledge Level and, further, also subscribe to the knowledge-level assertion of Clancey that classification problem solving involves the trivalent processes of data abstraction, heuristic match, and refinement to a problem of “...characterization of a particular case into one of a set of pre-enumerated possibilities” [13]. Further suppose that the target problem can be characterized as having a set of pre-enumerated answers. Utilizing the Knowledge Level, we assume that the processes of data abstraction, heuristic match, and refinement should exist in the Knowledge Level solution somewhere, but that is all we know. This argument that the KL as described by Newell is incomplete is more fully developed in [19].

Dietterich points out the same problem with using the Knowledge Level. In a clever example of a “perfect chess player” [20], Dietterich shows that the Knowledge Level provides a way of discussing what the chess playing agent knows, i.e., the knowledge the player has; but that the Knowledge Level gives us no clue about how to start building a chess playing agent. Indeed, although we can discuss the knowledge the agent has for any particular move, implementing such a “perfect chess player” would be infeasible. Although Dietterich is chiefly concerned with the “how to build it” issue, the “what predictions would I make” issue is inherent in his example.

The major thrust of Newell’s arguments is to move away from implementation level details toward a deeper understanding of problem solving. We argue that the Knowledge Level as it stands is incomplete due to its lack of a predictive component. The question is how can we prescribe an overall problem solving scheme to make predictive statements, and at the same time avoid implementation details? The short answer is that the way of prescribing the problem solving must be stated in terms appropriate to the problem solving activity itself.

0.2.4. Background Synthesis

Results of first generation, unitary problem solving approaches led some researchers (Breuker & Wielinga, Chandrasekaran, McDermott, Steels, ...) to develop representational approaches which were more direct embodiments of the domain knowledge that they sought to embed in their problem solvers. At the highest level, each developed domain and control *vocabularies* specialized for some specified set of problem solving methods. In addition to very practical concerns such as knowledge acquisition, this “task specific architectures” approach emphasized the need to develop families of implementation languages tailored for specific problem solving situations and available knowledge sources. Along a seemingly different track, Newell proposed the separation of analysis of knowledge systems from implementation concerns altogether.

In fact, these two thrusts have a common denominator: the desire to analyze problem solving in “natural” terms. For Newell, this term “natural” first meant pure knowledge-level terms,with no symbols to be used at all. For the TSA schools, “natural” meant in a representation and control vocabulary tailored for the task being modeled.

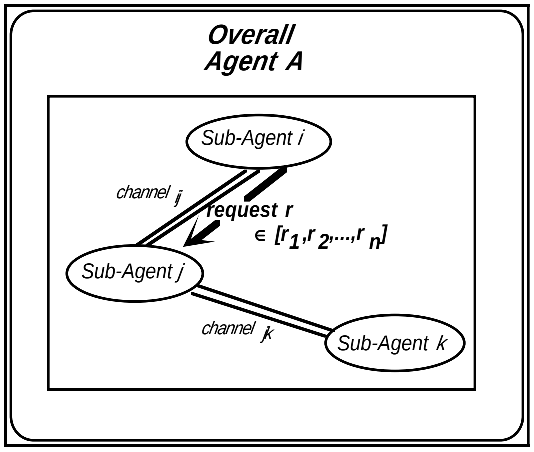
0.3. The Knowledge Level Architecture

In [19],Sticklen proposed an extension to the Knowledge Level Hypothesis (KLH) of Newell. This extension, the Knowledge Level Architecture Hypothesis (KLAH), is in fact a generalization of the TSA viewpoint.

> ***Knowledge Level Architecture Hypothesis (KLAH):*** If a problem solving agent may be decomposed into the cooperative efforts of a number of sub-agents, the larger agent can be understood at the Knowledge Level by giving a Knowledge Level description of the sub-agents and specifying the architecture the composition follows.

Each composed agent, taken as a complete entity, can be analyzed in accord with Newell’s view; each has components of goals, a body of knowledge, and primitive transducers, and each obeys the principle of rationality.

Each primitive agent also has goals, a body of knowledge, and primitive transducers that are shared with the composed agent. Here, too, the primitive agents obey the law of rationality. The only difference so far is that more primitive agents, which together composed a higher-level agent, and share primitive transducers. And further that these transducers appear to belong to the agent from the agent’s perspective.



The manner in which the primitive agents are allowed to interact with one another is **restricted** by an imposed **Knowledge Level Architecture**. This architecture is defined by two ingredients. First, the Knowledge Level Architecture must fix the paths of communication along which the sub-agents agents may interact. This is a way of setting the structural relationship between the sub-agents. Put another way, decomposing the agent to sub-agents and fixing the communication paths (taking both steps together) is a way of *organizing* the knowledge of the agent.

Second, the Knowledge Level Architecture must fix the message protocols which the sub-agents follow to communicate with one another. Assuming that sub-agents do not take independent action but rather act only when requested to, fixing the content of communications between the sub-agents is a way of expressing the goals of the sub-agents. The elements of the Knowledge Level Architecture are shown graphically below.

The hypothesis of a Knowledge Level Architecture should be taken as an adjunct to the Knowledge Level hypothesis itself; clearly if no such thing as a Knowledge Level exists, it does not have an architecture. If, however, there is a distinct Knowledge Level of the type described by Newell, then whenever the agent can be decomposed to sub-agents the Knowledge Level Architecture Hypothesis provides a framework for interpreting the actions of a composite agent by specifying the interaction of an ensemble of sub-agents.

Newell argues that the usefulness of the Knowledge Level lies in predicting and understanding behavior without having an operational model of the processing actually being done by the agent. The utility is basically to enable an implementation-free description of a problem solving agent. The Knowledge Level Architecture allows the same sort of implementation-free description while demanding an account of how the agent functions *at the Knowledge Level*. The central issue here is what we mean by “implementation-free.” Newell’s original notion of the Knowledge Level limits description of how a problem solving agent functions by disallowing any discussion of problem solving control. Our extension, on the other hand, allows discussion of such questions, *but only in the vocabulary of knowledge organization and control.*

The vocabulary of the KLA (subagents, communication paths, communication protocols) is used to analyze a given problem solver. To the extent the analysis is successful, a particular Knowledge Level architecture will be produced, and this will be tantamount to producing a task specific architecture for performing the stated task.

It is important to distinguish three levels of description for knowledge systems:

- ***The individual problem solver:*** The functioning MYCIN system is an example at this level. The vocabulary used for discussion here is the set of terms necessary in the domain of the problem solver; e.g., for MYCIN, the terms would include strep infection, patient fever, etc.
- ***The problem solving type:*** MYCIN performs a type of classification problem solving. But the utility of classification is much more general. The proper epistemic terms to use at this level are the primitives for classification itself; i.e., for any system that is undertaking classification. For example, all classification systems have categories around which final

answers are framed. Thus a necessary primitive for properly representing classification must be the classificatory category.

- ***The Knowledge Level Architecture:*** At this level, for agents which can be decomposed to an ensemble of cooperating sub-agents, we set the terms which will be used to describe the problem solving types expressed at the next lower level. These terms are in three categories: the principles for sub-agent decomposition, the organizational principles of the sub-agents (i.e., the communication channels), and the repertoire of messages which the sub-agents will understand. Note that the KLA sets the vocabulary for the description of *control* in the general problem solving types at the next lower level (the problem solving type level.

0.4. Link Between KBS Theory and KBS Application

There are two broad traditions for theory testing: mathematical proof and experimental testing. Some segments of AI tend more towards the mathematical-proof variant on theory testing, for example the logic school. But for Knowledge-Based Systems, the TSA viewpoint in general, and the KLAH provide a footing for the development of a experimental paradigm for theory testing.

Scientific theories must accomplish two goals. First, theory must account for known phenomena. Second, theory must make verifiable predictions about as yet unobserved phenomena. These criteria for scientific theory have been suggested by many philosophers of science, and have broad acceptance within the scientific community itself. For example, a widely used definition of science due to Conant is

> Science is an interconnected series of concepts and conceptual schemes that have developed as a results of experimentation and observation and are *fruitful* of further experimentation and observation. [21, page 25]

The “fruitful” part of Conant’s definition is aimed clearly at prediction. One way for an AI theory to be predictive is for it to have a prescriptive component, where our sense of “prescription” is that the theory offers guidance in the construction of problem solving systems in the domain of the theory.

Suppose that we have an intelligent agent A for which we hold a theory T. Following the above discussion, T should have two characteristics.

- First, given a known behavior B of A, T should explain why A took the action B as opposed to other possible actions. For example following the observation of an agent’s behavior, the Knowledge Level Hypothesis allows us to explain that behavior by ascribing to the agent compatible goals and knowledge, i.e., goals and knowledge that lead to the observed behavior.

- Second, T must also allow us to make predictions about future problem solving behavior B' of A.

The first point above is obvious; the second requires more discussion. In science, when we say “Theory T predicts that B will be observed under conditions...,” then an attempt is usually made to verify the prediction as a *test* of the theory. Of course, this is precisely why the second, predictive element of theory is important. Without the requirement that theory should predict phenomena as yet unobserved, we run the risk of simply fitting theory to known observations.

Centering now on the need to make predictions about future behavior, the next task becomes indicating *how* such predictions can be made. Let us look to a well established science for inspiration: physics. For some theories of physical phenomena, prediction becomes a matter of working out a closed form solution to some mathematical expression. However, not all situations which physical theory address have closed form solutions. Consider, for example, the astrophysical theory of stellar structure. There are five relatively simple equations of state which describe the theory of various forces in action inside a star. There is in general no closed form solution to these equations. Thus in order to make useful predictions from the current state of their theory, astrophysicists use the equations of state as the basis for a time based numerical simulation. The results of this simulation are predictions of the physical behavior of stars over time. Note that there are two bodies of knowledge which the astrophysicist utilizes to make his predictions: knowledge of the physical theory as expressed mathematically, and knowledge of how to perform a numerical simulation based on such equations. The two types of knowledge are forged into a simulation *model* of a star that is then capable of prediction.

No one would confuse a simulation model of a star with the actual star — it would be hard to get enough energy out a simulation model to energize a planetary system! But suppose that our theory concerns itself with problem solving agents, and the phenomena we want to make predictions about is the behavior of a problem solving agent. If we construct a simulation model S of some problem solving agent A which is based on a theory T , and this model can successfully predict behavior B' of A, then in fact our simulation model S is itself a (constructed) problem solving agent, at least if one accepts the Turing Test as a basis for calling something a problem solving agent. In the realm of information systems, a simulator of a problem solving agent is itself a problem solving agent.

Which kind of prediction (direct or simulation) do/will problem solving theories support? The “do” part is straightforward — currently there do not exist theories of problem solving that make direct detailed predictions about future behavior. But we cannot say that there will *never* be such a theory, and hence a definitive answer cannot be offered for the “will” part of the question. There are a number of research programs currently under way that will produce predictions about problem solving behavior (for example, the SOAR project) but all are based on a simulation of a problem solving agent.

If theory of problem solving is simulation-based, then another way of viewing it is that it is a statement of *how to construct* an agent to undertake the problem solving. This is a result of a strong difference between theory-based simulations that are predictive in the physical sciences, and theory-based simulations that are predictive for problem solving phenomena.

And this brings us to the bottom line for this section. Although there is not conclusive proof that we cannot construct problem solving theory capable of “direct prediction” for problem solving, current experience leads us to believe that such theory would be at best very difficult to enunciate.

Typically, we associate a prescriptive set of directions for building an artifact with an engineering discipline, not with a science. In the case of theories of problem solving, we have argued that, in order for the theory to have the important criterion of the ability to predict phenomena, the theory will be a statement of how to build a simulator for problem solving. But that simulator itself will be a problem solving agent. Hence, in addition to being a scientific statement, the theory is also a prescriptive statement for building such agents. This argument sheds light on the confusion that is often seen in arguments about whether knowledge-based systems is an engineering discipline or a science: a good KBS theory is *both*.

0.5. Conclusion

In this report, we have described an extension to Newell’s Knowledge Level Hypothesis (KLH) called the Knowledge Level Architecture Hypothesis (KLAH). First described in [19], the KLAH is best understood as a generalization of the Task Specific Architectures view that has been gaining adherents over the last decade. A similar intuition was recently used by Van de Velde (of Steels group) to offer a different extension of Newell’s Knowledge Level. While we have proposed a decomposition of the monolithic agent of Newell to the cooperative efforts of a number of subagents connected together by our KLA, Van de Velde decomposed the “principle of rationality” of Newell into a general part (similar to Newell’s original) and to a second portion which would contain the control goals of particular tasks [22]. Even more recently, the SOAR community under Newell’s leadership has also developed a systems description level intermediate between the symbol level and the knowledge level [23], which addresses similar concerns to our KLA level of description.

We have described how the KLAH provides a framework in which particular TSAs may be viewed as Knowledge Level constructs. A particular TSA can then be used both as a prescription for building actual systems and as a statement of a problem solving theory. Once the TSA-blueprinted system is built, it may be used to predict future behavior — the TSA statement of theory is susceptible to experimental testing.

We started this report with a discussion of the shortcomings of the unitary problem solving techniques of the first generation of knowledge-based systems. Those shortcomings were typically traced to the representation of problem solving being carried out at too low a level, typically in a computation-universal framework. The bottom line of our argument for TSAs and for the KLAH framework for understanding TSAs is that by constraining what can be done in a given problem solver to targeted tasks, and doing this through limitations in representational primitives and control constructs, it is possible to forge a link between problem solving theory and problem solving application. The central facet of our argument is the *duality* for problem solving systems between a blueprint to build the system and a statement of the theory backing the problem solver.

This report owes much to the intellectual ferment and excitement of the Laboratory for AI Research at Ohio State University, and to all associated with that laboratory. In addition, recent discussions between the first author and Luc Steels have been helpful in settling these ideas. Comments of the editors, reviewers, and discussants of the original statement of the KLAH, which appeared in **JETAI**, were extremely useful in helping to clarify our proposal.

References

1. Buchanan, B. and E.H. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. 1984, Addison-Wesley.
2. Shortliffe, E.H., *Computer Based Medical Consultations: MYCIN*. 1976, Elsevier North Holland Inc.
3. Chandrasekaran, B. *Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks*. in *IJCAI-87*. 1987. Milan.
4. Duda, R.O. and J. Gaschnig, *Model Design in the Prospector Consultant System for Mineral Exploration*, in *Expert Systems in the Micro-Electronic Age*, D. Michie, Editor. 1979, Edinburgh University Press.
5. Bylander, T. and B. Chandrasekaran, *Generic Tasks for Knowledge-Based Reasoning: The 'Right' Level of Abstraction for Knowledge Acquisition.* Int. J. Man-Machine Studies, 1987. **26**(2): p. 231-243.
6. Gruber, T. and P. Cohen, *Design for Acquisition: Principles of Knowledge-System Design to Facilitate Knowledge Acquisition.* International Journal of Man-Machine Studies, 1987. **26**: p. 143-159.
7. Chandrasekaran, B. *Decomposition of Domain Knowledge into Knowledge Sources: The MDX Approach*. in *Proc. 4th Nat. Conf. Canadian Society for Computational Studies of Intelligence*. 1982.

8. Chandrasekaran, B., *Towards a Taxonomy of Problem-Solving Types.* AI Magazine, 1983. **4**(1): p. 9-17.

9. Chandrasekaran, B., *Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design.* IEEE Expert, 1986. p. 23-30.

10. Chandrasekaran, B., J.W. Smith, and J. Sticklen, *Deep Models and their Relation to Diagnosis,* in *Artificial Intelligence in Medicine,* Furukawa, Editor. 1989, Science Publishers: Amsterdam, Netherlands.

11. Clancey, W.J. *NEOMYCIN: Reconfiguring a Rule-Based Expert System for Application To Teaching.* in *Proceedings of IJCAI 7*. 1981.

12. Clancey, W.J. *Advantages of Abstract Control Knowledge in Expert System Design*. in *Proceedings of AAAI*. 1983.

13. Clancey, W.J. *Classification Problem Solving*. in *Proceedings of the AAAI*. 1984.

14. Clancey, W.J., *Representing Control Knowledge as Abstract Tasks and Metarules.* 1985, Stanford University: Palo Alto.

15. McDermott, J., *Preliminary Steps Toward a Taxonomy of Problem-Solving Methods,* in *Automating Knowledge Acquisitionfor Expert Systems,* S. Marcus, Editor. 1988, Kluver Academic Publishers: Boston. p. 225-255.

16. Steels, L., *The Components of Expertise.* AI Magazine, 1990. **Summer, 1990**.

17. Breuker, J. and B. Wielinga, *Models of Expertise in Knowledge Acquisition,* in *Topics in Epxert Systems Design: Methodologies and Tools,* G. Guida and C. Tasso, Editor. 1989, North Holland Publishing Company: Amsterdam.

18. Newell, A., *The Knowledge Level.* AI Magazine, 1982. **Summer**: p. 1-19.

19. Sticklen, J., *Problem Solving Architectures at the Knowledge Level.* Journal of Experimental and Theoretical Artificial Intelligence, 1989. **1**: p. 1-52.

20. Dietterich, T.G., *Learning at the Knowledge Level.* Machine Learning, 1986. **1**: p. 287-316.

21. Conant, J.B., *Science and Common Sense*. 1951, Yale University Press.

22. Van de Velde, W. *Tractable Rationality at the Knowledge Level*. in *Artificial Intelligence and Simulation of Behaviour (AISB)*. 1991. Leeds, GB: Springer-Verlag.

23. Newell, A., *et al.*, *Formulating the problem space computational model,* in *Carnegie-Mellon Computer Science: A 25-year Commemorative Reading,* R.F. Rashid, Editor. 1991, ACM-Press, Addison-Wesley.