An Overview of Limits to Knowledge Level-B Modeling (and KADS)*

Tim Menzies †

June 14, 1995

Abstract

Despite the current enthusiasm for knowledge level-B modeling in general and KADS in particular, we find their exists little empirical proof of the utility of this approach. In this paper, we will review the available evidence to conclude that alternative, less abstract, approaches are arguably better than KADS.

1 Introduction

In the 1970s and early 1980s, several high-profile expert system successes were documented: e.g. MYCIN [57], CASNET [58], PROSPECTOR [8, 17], and XCON [2]. However, despite careful attempts to generalise this work (e.g. [53]), expert systems construction remained a somewhat hit-and-miss process. By the end of the 1980s, it was recognised that our design concepts for knowledge-based systems were incomplete [7].

A new expert system design approach (which has come to dominant the knowledge acquisition (KA) field) is the the search for reusable abstract domain-independent problem-solving strategies. We call this approach $\mathcal{KL}_B$ since it is a variant of Newell's *knowledge level* (KL) modeling approach [35, 37, 36]. KADS is a $\mathcal{KL}_B$ variant used by many commercial expert systems practioners[1]. Wielinga *et. al.* note that, as of 1992, KADS has been used in some 40-to 50 KBS projects, 17 of which are described in published papers [59]. Amongst AI-94 participants, interest in KADS was high.

Despite the current enthusiasm for $\mathcal{KL}_B$ and KADS, we find their exists little empirical proof of the utility of this approach. If we review the available evidence, we find that alternative, less abstract, approaches are arguably more productive than KADS. We therefore believe that we should retreat from high-level $\mathcal{KL}_B$ approaches to techniques based on a much lower level of abstraction. We know of no publication that summarises these results. Hence, this article. Due to space limitations, this article will be an overview only (a longer version is in preparation).

2 A Tutorial on Knowledge-Level Modeling

In Newell's KL approach, intelligence is modeled as a search for appropriate *operators* that convert some *current state* to a *goal state*. Domain-specific knowledge as used to select the operators according to *the principle of rationality*; i.e. an intelligent agent will select an operator which its knowledge tells it will lead the achievement of some of its goals. If implemented, this KL is built on top of a *symbol-level* containing data structures, algorithms, etc. However, to a KL agent, these sub-cognitive symbol-level constructs are the tools used "sub-consciously" as it performs its KL processing [35].

*Submitted to the Australian AI '95 conference.

†Department of Software Development, Monash University, Caulfield East, Melbourne, VIC.; timm@insect.sd.monash.edu.au

[1] See http://www.swi.psy.uva.nl/projects/CommonKADS/Reports.html for on-line versions of the KADS documentation. See [55] for a detailed text on KADS. See [26] for a good tutorial introduction to KADS.

Newell's subsequent exploration of the KL lead to a general rule-based language called SOAR [49] which was the basis for the problem-space computational model (PSCM) [61]. Programming SOAR using the PSCM involves the consideration of multiple, nested problem spaces. Whenever a "don't know what to do" state is reached, a new problem space is forked to solve that problem. Newell concluded that the PSCM was the bridge between SOAR and true KL modeling [37, 36].

We distinguish between PSCM (which we term $\mathcal{KL}_A$) and $\mathcal{KL}_B$, a KL-modeling variant which groups together a set of authors who argue for basically the same technique; i.e. Clancey's model construction operators [11], Steels' components of expertise [51], Chandrasekaran's task analysis, SPARK/ BURN/ FIREFIGHTER [27] and KADS [59][2]. The fundamental premise of $\mathcal{KL}_B$ is that a knowledge base should be divided into domain-specific facts and domain-independent problem solving methods. For example, Clancey argues that knowledge engineering should separate heuristics like Figure 1 into domain-specific assertions about the terminology (see Figure 2), re-usable problem solving strategies (see Figure 3) and true domain-specific heuristics (see Figure 4) [11]. Such problem-solving strategies are implicit in $\mathcal{KL}_A$. The observation that a PSCM system is performing (e.g.) classification is a user-interpretation of a lower-level inference (operator selection over a problem space traversal) [61].

if the infection in meningitis *and*
the type of infection in bacterial *and*
the patient has undergone surgery *and*
the patient has undergone neurosurgery *and*
the neurosurgery-time was < 2 months ago *and*
the patient received a ventricular-urethral-shunt
then infection = e.coli (.8) *or* klebsiella (.75)

Figure 1: A rule. Domain terms are underlined.

subtype(meningitis, bacteriaMenigitis).
subtype(bacteriaMenigitis, eColi).
subtype(bacteriaMenigitis, klebsiella).
subsumes(surgery, neurosurgery).
subsumes(neurosurgery, recentNeurosurgery).
subsumes(recentNeurosurgery, ventricularUrethralShunt).
causalEvidence(bacteriaMenigitis, exposure).
circumstantialEvidence(bacteriaMenigitis, neurosurgery).

Figure 2: Domain knowledge from Figure 1.

Strategy	*Description*
exploreAndRefine	Explore super-types before sub-types.
findOut	If an hypothesis is subsumed by other findings which are not present in this case then that hypothesis is wrong.
testHypothesis	Test causal connections before mere circumstantial evidence.

Figure 3: Problem solving strategies from Figure 1.

In the $\mathcal{KL}_B$ view (which we believe is overly-optimistic), knowledge engineering becomes a structured search for an appropriate problem solving strategy. Once a strategy is found or developed, then KA

[2] See the *Related Work* section of [59] for a discussion of the differences in these techniques

if the patient received a ventricular-urethral-shunt
then infection = e.coli (.8) *or* klebsiella (.75)

Figure 4: The true heuristic contained in Figure 1.

becomes a process of filling in the details required to implement that strategy. Libraries of problem solving strategies become a productivity tool for building a wide-variety of expert systems. All known problem solving strategies (e.g. prediction, monitoring, diagnosis, cover & differentiate, propose & revise, planning, design, verification, assessment) are really combinations of a small number (say, ≤ 20) of reusable inference subroutines (e.g. instantiate, generalise, abstract, specify, select, assign-value, compute, compare, match, assemble, decompose, transform). New problem strategies can be quickly built out of these lower-level inference primitives. Knowledge engineers can now separate their analysis work from their design and coding work. Analysis is the process of generating abstract problem solving strategies. Design and coding is a process of implementing these strategies.

$\mathcal{KL}_B$ is not a theory of human cognition. The current public line in the KA community is that knowledge bases are inaccurate approximate surrogate models of reality [14, 20]. Gone are the days of "expertise-transfer" where KA workers believed they were "mining the jewels in the expert's head" (to use Feigenbaum's poetic phrase [19]). While some knowledge representation theorists still make occasional claims that their knowledge representation theory has some psychological basis (e.g. portions of the $\mathcal{KL}_A$ research reported in [49, page xxvii]), the official public line is that representations are models/surrogates only [41].

3 Assessment of $\mathcal{KL}_B$

3.1 Evidence For $\mathcal{KL}_B$?

Marques *et. al.* report significantly reduced development times for expert systems using a library of 13 reusable inference subroutines (eliminate, schedule, present, monitor, transform-01, transform-02, compare-01, compare-02, translate-01, translate-02, classify, select, dialog-mgr) in the SPARK/ BURN/ FIREFIGHTER (SBF) environment. In the nine studied applications, development times changed from one to 17 days (using SBF) to 63 to 250 days (without using SBF) [27]. To our knowledge, this is the largest documented evidence of productivity gains in any software approach (be it knowledge based or otherwise). However:

- In the SBF experiments, the mappings between a graphical language for expressing the specification to the library of inference subroutines was hand-coded. Techniques to assist/automate this mapping process have yet to evolve. The crucial test for SBF is how well this system ports to domains outside of the areas it was initially developed for. We find it significant that the SBF group made no entry to the Sisyphus-1 or Sisyphus-2 KA project (Sisyphus is an attempt by the international KA community to define reproducible KA experiments [26]). Most other $\mathcal{KL}_B$ groups offered solutions to the 1991, 1992, and 1994 Sisyphus rounds. There was even a $\mathcal{KL}_A$ contribution to the 1994 Sisyphus round [60]. If SBF was a general productivity tool, then it should have been able to quickly build Sisyphus solutions.

Clancey's classic *Heuristic Classification* paper [9] offered a unified retrospective view on numerous, seemingly different, expert systems. Similar (but smaller) studies (e.g. [1, 26]) suggest that $\mathcal{KL}_B$ can retrospectively clarify historical expert systems design issue. However:

- Two important issues remain outstanding: (i) that $\mathcal{KL}_B$ can simplify a *current* developing design; and (ii) that $\mathcal{KL}_B$ is a better design methodology than other approaches. Motta & Zdrahal discuss various $\mathcal{KL}_B$ approaches using their special knowledge of a symbol-level procedure called constraint satisfaction [62]. We find their lower level more insightful into the construction process than the less-detailed, high-level $\mathcal{KL}_B$ approach. Further, this low-level view of a problem can find errors that experienced $\mathcal{KL}_B$ practioners cannot. For example, one declarative translation of the procedures in

the Sisyphus-2 specification blurred the distinction between hard constraints (which must not be violated) and soft constraints (which can be optionally violated) [34].

The separation of problem solving methods from domain assertions and true heuristics (e.g. Figure 1 to Figure 4) can optimise knowledge base processing. Clancey reports that after a $\mathcal{KL}_B$ analysis of 176 MYCIN rules, he could generate a new knowledge base where 80% of the rules had only a single condition. Further, the problem solving strategies removed all uncontrolled backtracking [11]. However, this result has not been reported elsewhere.

3.2 Are Problem Solving Methods Reusable?

It is not clear that the problem solving methods found by $\mathcal{KL}_B$ are truly reusable. Between the various camps of $\mathcal{KL}_B$ researchers, there is little agreement on the details of the problem solving methods. Contrast the list of inference sub-routines from KADS [59] and SPARK/ BURN/ FIREFIGHTER [27] (termed "knowledge sources' and "mechanism" respectively). While there is some overlap, the lists are different. Also, the number and nature of the problem solving methods is not fixed. Often when a domain is analysed using $\mathcal{KL}_B$, a new method is induced [26]. Further, different interpretations exist of the same method. For example, the problem solving method proposed by Bredeweg [5] for prediction via qualitative reasoning is different to the qualitative prediction method proposed by Tansley & Hayball [55].

3.3 Do We Need a Model?

Surprisingly, having no model of a problem solving method may be better than having one. Corbridge *et. al.* report a counter-intuitive experimental result [13] in which subjects working with no models extracted more knowledge from an expert dialogue than subjects working with a KADS model. Far from challenging this result, the $\mathcal{KL}_B$ community is now exploring empirical methods for exploring its approach in the Sisyphus-3 project.

Compton reports experiments where experts could fix faulty rules using an *unless* patch attached at the end of a rule condition. Patches are themselves rules which can be recursively patched. Experts can never re-organise the tree; they can only continue to patch their patches. These *ripple-down-rule* (RDR) trees are a very low-level representation. Rules cannot assert facts that other rules can use. In no way can a RDR tree be called a model in anything like a $\mathcal{KL}_B$ sense. Yet this low-level model-less approach has produced large working expert systems in routine daily use. For example, the PIERS system at St. Vincent's Hospital, Sydney, models 20% of human biochemistry sufficiently well to make diagnoses that are 99% accurate [47]. RDR has succeeded in domains where previous attempts, based on much higher-level constructs, never made it out of the prototype stage [43]. Further, while large expert systems are notoriously hard to maintain [15], the no-model approach of RDR have never encountered maintenance problems. System development blends seamlessly with system maintenance since the only activity that the RDR interface permits is patching faulty rules in the context of the last error. For a 2000-rule RDR system, maintenance is very simple (a total of a few minutes each day).

Compton argues that his process of "patching in the context of error" is a more realistic model of human reasoning than assuming that a human analyst will behave in a perfectly rational way to create some initial correct design [12]. This line is perused further in the *situated cognition* literature where it is claimed that it is folly to use context-independent symbolic assertions to model human reasoning [3, 10, 28, 50]. While many symbolic AI researchers dispute this claim (e.g. [33, 56]), it has some following within the $\mathcal{KL}_B$ community (e.g. Steels [52] and Clancey [10]).

Our view is not an extreme situated cognition position. We do not reject $\mathcal{KL}_B$ and KADS out of a disdain for symbolic logics. Like Poole [46], we believe that we can use logics to represent commonsense reasoning in general and context-dependent reasoning in particular, just as long as we don't use deductive logics. Our preferred approach is *abductive* (see below).

3.4 Is Symbol-Level Modeling Better?

We believe that the high-level abstract view provided by $\mathcal{KL}_B$ obscures important distinctions at the symbol-level. To be fair, the KADS community notes that some overlap exists between problem solving methods: Wielinga *et. al.* note that both monitoring and systematic diagnosis share some processing [59].

However, having noted that some low-level similarities exist between the KL distinctions that they propose, they do not take the next step and simplify their distinctions accordingly. We believe that *abduction* is a symbol-level inference procedure that can unify a wide variety of KL tasks. Consider a dependency and-or graph with invariants $\mathcal{I}$, edges $\mathcal{E}$ and literals stored in vertices $\mathcal{V}$. Abduction is the generation of maximal consistent (with respect to $\mathcal{I}$) worlds $\mathcal{W}$ ($\mathcal{W}_x \subseteq \mathcal{E}$) that contain some subset of the desired goals $\mathcal{G}$ using some subset of known inputs $\mathcal{IN}$ ($\mathcal{IN} \subseteq \mathcal{V}, \mathcal{G} \subseteq \mathcal{V}$). In the case where multiple worlds can be generated, a $\mathcal{BEST}$ operator selects the preferred world(s). By choosing appropriate $\mathcal{IN}$, $\mathcal{G}$, and $\mathcal{BEST}$, a range of knowledge base tasks can be implemented:

- Parsimonious set-covering diagnosis [48] uses a $\mathcal{BEST}$ that favors worlds that explain the most things, with the smallest number of diseases (i.e. maximise $\mathcal{W}_x \cap \mathcal{G}$ and minimise $\mathcal{W}_x \cap \mathcal{IN}$).
- Conceptually, $\mathcal{BEST}$ is applied after the worlds are generated. However, certain $\mathcal{BEST}$s can be applied during generation. For example, if it is known that $\mathcal{BEST}$ will favour the worlds with smallest path sizes between inputs and goals, then a beam-search style $\mathcal{BEST}$ operator could cull excessively long proofs within the generation process.
- Case-based reasoning uses a $\mathcal{BEST}$ that favours worlds that use the most number of edges that have been used in prior acceptable solutions [25].
- Explanation uses a $\mathcal{BEST}$ that favours the worlds with the most number of edges that model processes which are familiar to the user [42] and which include the largest subset of $\mathcal{G}$ ($\mathcal{IN}$ is set to the possible inputs).
- Prediction is implemented by calling the abductive inference with $\mathcal{G} \subseteq \mathcal{V} - \mathcal{IN}$; i.e. find all vertices we can reach from the inputs. This is a non-naive implementation of prediction since mutually exclusive predictions will be found in different worlds. Note that in the special case where $\mathcal{IN}$ are all root vertices in the graph and $\mathcal{G} = \mathcal{V} - \mathcal{IN}$, then our abductive system will compute ATMS-style [16] total envisionments; i.e. all possible consistent worlds that are extractable from the theory. A more efficient case is that $\mathcal{G} \subset \mathcal{V} - \mathcal{I}$; i.e. some *interesting subset* of the vertices have been identified as possible outputs.
- Classification is just a special case of prediction. Given a dependency graph that connects classes to their attributes and their super-classes, $\mathcal{G}$ is just the vertices that are classes. $\mathcal{BEST}$ could also favours the worlds that include the more general classes [44].
- Validation uses a $\mathcal{BEST}$ that favours the largest number of covered outputs (i.e. maximise $\mathcal{G} \cap \mathcal{W}_x$) where $\mathcal{G}$ and $\mathcal{IN}$ come from a library of known behaviour of the thing being modeled [31],

Elsewhere, we discuss how to use abduction for planning, monitoring, qualitative reasoning, verification, multiple-expert knowledge acquisition, explanation, single-user decision support systems and multiple-user decision support systems [29]. We also believe that abduction can model certain interesting features of human cognition [30]. Other authors have discussed the use of abduction for for natural-language processing [38], design [45], visual pattern recognition [46], analogical reasoning [18], financial reasoning [22] and machine learning [23].

We suspect that the $\mathcal{KL}_B$ community has nearly discovering abduction several times. For example, we view Clancey's distinction between heuristic classification and heuristic construction [9] as the difference between abduction with no invariants (where every combination of proof is legal) and abduction with invariants (where proofs compete with each other). Also, Brueker's recent discussion of *components of solutions* for expert systems [6] sounds to us like three recursive calls to a single inference procedure. Brueker claims that different problems solving methods can be interfaced to each other if we recognise that they all contain the same four components of solutions: an *argument structure* which is extracted from a *conclusion* which is in turn extracted from a *case model* which is in turn extracted from a *generic domain model*. Note that, in all cases, each sub-component is generated by extracting a relevant subset of some background theory to generate a new theory (i.e. abduction).

4 If not KADS, Then What?

We acknowledge that the techniques we are proposing to replace $\mathcal{KL}_B$ (RDR, constraint satisfaction, abduction) are not as developed and documented as $\mathcal{KL}_B$. Hence, they may not be practical for commercial practioners. Our advice to such practioners is to use Boehm's spiral model [4] and Jacobsen's use cases [24] to control the evolutionary development of expert systems. A graph of % use-cases implemented vs time is a powerful management tool for tracking developments that require prototyping. We suggest that a "thin path" be found through the design that allows some subset of the use cases to be executed at an early stage of the development. This gives business users the feedback required for them to mature their specification. The knowledge engineer's task is then to widen the "thin path" to cover all the planned module.

An initial prototype could be built using RDR trees. Hopefully, this prototype will grow into your full application. If not, then more conventional software development techniques will be required. Try to avoid pre-maturely hard-wiring the control of system. The blackboard approach is a good technique for avoiding premature control structures [40, 39]. Also, try to set up your system such the same routine can be used to compute outputs from inputs inputs from outputs and visa versa. We have no definite recommendation for implementation languages, but have a preference for Prolog for the inference engine and Smalltalk for the interface.

5 Conclusion

Our main complaint against $\mathcal{KL}_B$ is the that it is *advocacy* research rather than *empirical* research. Advocacy research proceeds by researchers expressing their belief that a certain technique is a worthy thing. Empirical research proceeds by researchers looking for quantifiable evidence that some advocated technique actually works in practice. We have argued elsewhere that conventional software engineering has insufficient empirical research [32]. Here, we argue that knowledge engineering has the same problem since, when we look for evidence of the utility of $\mathcal{KL}_B$, don't find much. Empirical and theoretically, we find more support for the use of tools at a lower-level of abstraction than $\mathcal{KL}_B$ (the RDR experiments, our abductive work, and the studies by Motta & Zdrahal).

Our criticisms of $\mathcal{KL}_B$ do not imply a criticism of $\mathcal{KL}_A$. When we look into the details of PSCM-SOAR, we find that, like our abductive proposal, PSCM-SOAR uses a low-level uniform structure for KBS design. The difference between $\mathcal{KL}_A$ and abduction is that the forward-chaining of PSCM-SOAR is best suited to single-world deductive reasoning (though some work on abduction has been done in the $\mathcal{KL}_A$ paradigm [54]).

We take care to distinguish Clancey's $\mathcal{KL}_B$ work from other $\mathcal{KL}_B$ research. Based on a careful reverse engineering of previously successful designs, Clancey's work is founded on documented experience. The same can not be said about other $\mathcal{KL}_B$ work. For example, many of the problem solving methods listed in Tansley & Hayball were created especially for that book by the authors from their own undocumented sources [55, page 260]. Further, Clancey is honest enough to publish his doubts of the symbolic paradigm [10].

In challenging the utility of $\mathcal{KL}_B$, we are also challenging a premise at the heart of modern software engineering; i.e. abstractions are a good thing and higher-level abstractions are better than lower-level abstractions. The notion of abstract design patterns is a pervasive concept and can see a similar idea in the object-oriented community [21]. Extrapolating the KADS experience to this object-oriented software patterns research, we doubt the long-term viability of this approach. Knowledge and software engineering, we argue, should be based on lower-level abstractions than what is seen in current practice.

References

[1] H. Akkermans, F. van Harmelen, G. Schreiber, and B. Wielinga. A formalisation of knowledge-level models for knowledge acquisition. In J. Balder and H. Akkermans, editors, *Formal Methods for Knowledge Modeling in the CommonKADS Methodology: A Compilation (KADS-EE/T1.2/TR/ECN/014/1.0)*, pages 53–90. Netherlands Energy Research Foundation, 1992.

[2] J. Bachant and J. McDermott. R1 revisited: Four years in the trenches. *AI Magazine*, pages 21–32, Fall 1984.

[3] L. Birnbaum. Rigor mortis: A response to Nilsson's "Logic and Artificial intelligence". *Artificial Intelligence*, 47:57–77, 1991.

[4] B. Boehm. A spiral model of software development and enhancement. *Software Engineering Notes*, 11(4):22, 1986.

[5] B. Bredeweg. *Expertise in Qualitative Prediction of Behaviour*. PhD thesis, University of Amsterdam, 1992.

[6] J. Breuker. Components of problem solving and types of problems. In *8th European Knowledge Acquisition Workshop, EKAW '94*, pages 118–136, 1994.

[7] B.G. Buchanan and R.G. Smith. Fundamentals of expert systems. In P.R. Cohen A. Barr and E.A. Feigenbaum, editors, *The Handbook of Artificial Intelligence, Volume 4*, volume 4, pages 149–192. Addison-Wesley, 1989.

[8] A.N. Campbell, V.F. Hollister, R.O. Duda, and P.E. Hart. Recognition of a Hidden Material Deposit by and Artificially Intelligent Program. *Science*, 217:927–929, 3 September 1982.

[9] W. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[10] W. Clancey. Situated Action: A Neuropsychological Interpretation Response to Vera and Simon. *Cognitive Science*, 17:87–116, 1993.

[11] W.J. Clancey. Model construction operators. *Artificial Intelligence*, 53:1–115, 1992.

[12] P.J. Compton and R. Jansen. A philosophical basis for knowledge acquisition. *Knowledge Acquisition*, 2:241–257, 1990.

[13] C. Corbridge, N.P. Major, and N.R. Shadbolt. Models exposed: An empirical study. In *Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge Based Systems*, 1995.

[14] R. Davis, H. Shrobe, and P. Szolovits. What is a knowledge representation? *AI Magazine*, pages 17–33, Spring 1993.

[15] A. Van de Brug, J. Bachant, and J. McDermott. The taming of R1. *IEEE Expert*, pages 33–39, Fall 1986.

[16] J. DeKleer. An assumption-based TMS. *Artificial Intelligence*, 28:163–196, 1986.

[17] R.O. Duda, P.E. Hart, and R. Reboh. Letter to the editor. *Artificial Intelligence*, 26:359–360, 1985.

[18] B Falkenhainer. Abduction as similarity-driven explanation. In P. O'Rourke, editor, *Working Notes of the 1990 Spring Symposium on Automated Abduction*, pages 135–139, 1990.

[19] E. Feigenbaum and P. McCorduck. *The Fifth Generation*. Addison-Wesley, 1983.

[20] B. Gaines. AAAI 1992 spring symposium series reports: Cognitive aspects of knowledge acquisition. *AI Magazine*, page 24, Fall 1992.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[22] W. Hamscher. Explaining unexpected financial results. In P. O'Rourke, editor, *AAAI Spring Symposium on Automated Abduction*, pages 96–100, 1990.

[23] K. Hirata. A classification of abduction: Abduction for logic programming. In *Proceedings of the Fourteenth International Machine Learning Workshop, ML-14*, page 16, 1994.

[24] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.

[25] D.B. Leake. Focusing construction and selection of abductive hypotheses. In *IJCAI '93*, pages 24–29, 1993.

[26] M. Linster and M. Musen. Use of KADS to create a conceptual model of the ONCOCIN task. *Knowledge Acquisition*, 4:55–88, 1 1992.

[27] D. Marques, G. Dallemagne, G. Kliner, J. McDermott, and D. Tung. Easy programming: Empowering people to build their own applications. *IEEE Expert*, pages 16–29, June 1992.

[28] D. McDermott. A critique of pure reason. *Computational Intelligence*, 3:151–160, 1987.

[29] T.J. Menzies. *Principles for Generalised Testing of Knowledge Bases*. PhD thesis, University of New South Wales, 1995.

[30] T.J. Menzies. Situated semantics is a side-effect of the computational complexity of abduction. In *Australian Cognitive Science Society, 3rd Conference*, 1995. Available from //www.sd.monash.edu.au/ ~timm/pub/docs/papers.html.

[31] T.J. Menzies and W. Gambetta. Exhaustive abduction: A practical model validation tool. In *ECAI '94 Workshop on Validation of Knowledge-Based Systems*, 1994. Available from //www.sd.monash.edu.au/ ~timm/pub/docs/papers.html.

[32] T.J. Menzies and P. Haynes. The methodologies of methodologies; or, evaluating current methodologies: Why and how. In *Tools Pacific '94*, pages 83–92. Prentice-Hall, 1994.

[33] M. Minsky. Society of Mind: A response to four reviews. *Artificial Intelligence*, 48:371–396, 1991.

[34] E. Motta and Z. Zdrahal. The trouble with what: Issues in method-independent task specifications. In *Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop Banff, Canada*, 1995.

[35] A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[36] A. Newell. Reflections on the knowledge level. *Artificial Intelligence*, 59:31–38, Feburary 1993.

[37] A. Newell, G.R. Yost, J.E Laird, P.S. Rosenbloom, and E. Altmann. Formulating the problem space computational model. In P.S. Rosenbloom, J.E. Laird, and A. Newell, editors, *The Soar Papers*, volume 2, pages 1321–1359. MIT Press, 1991.

[38] H.T. Ng and R.J. Mooney. The role of coherence in constructing and evaluating abductive explanations. In *Working Notes of the 1990 Spring Symposium on Automated Abduction*, volume TR 90-32, pages 13–17, 1990.

[39] H.P. Nii. Blackboard systems: Blackboard application systems, blackboard systems from a knowledge engineering perspective. *AI Magazine*, pages 82–106, August 1986.

[40] H.P. Nii. Blackboard systems: The blackboard model for problem solving and the evolution of blackboard architectures. *AI Magazine*, pages 38–53, Summer 1986.

[41] K. O'Hara and N. Shadbolt. AI models as a variety of psychological explanation. In *IJCAI '93*, volume 1, pages 188–193, 1993.

[42] C.L. Paris. The use of explicit user models in a generation system for tailoring answers to the user's level of expertise. In A. Kobsa and W. Wahlster, editors, *User Models in Dialog Systems*, pages 200–232. Springer-Verlag, 1989.

[43] R.S. Patil, P. Szolovitis, and W.B Schwartz. Causal understanding of patient illness in medical diagnosis. In *IJCAI '81*, pages 893–899, 1981.

[44] D. Poole. On the comparison of theories: Preferring the most specific explanation. In *IJCAI '85*, pages 144–147, 1985.

[45] D. Poole. Hypo-deductive reasoning for abduction, default reasoning, and design. In P. O'Rourke, editor, *Working Notes of the 1990 Spring Symposium on Automated Abduction.*, volume TR 90-32, pages 106–110, 1990.

[46] D. Poole. A methodology for using a default and abductive reasoning system. *International Journal of Intelligent Systems*, 5:521–548, 1990.

[47] P. Preston, G. Edwards, and P. Compton. A 1600 rule expert system without knowledge engineers. In J. Leibowitz, editor, *Second World Congress on Expert Systems*, 1993.

[48] J. Reggia, D.S. Nau, and P.Y Wang. Diagnostic expert systems based on a set covering model. *Int. J. of Man-Machine Studies*, 19(5):437–460, 1983.

[49] P.S. Rosenbloom, J.E. Laird, and A. Newell. *The SOAR Papers.* The MIT Press, 1993.

[50] J.R. Searle. Minds, brain, and programs. *The Behavioral and Brain Sciences*, 3:417–457, 1980.

[51] L. Steels. Components of expertise. *AI Magazine*, 11:29–49, 2 1990.

[52] L. Steels. How can we make further progress in knowledge acquisition? In R. Mizoguchi, H. Motoda, J. Boose, B. Gaines, and P. Compton, editors, *Proceedings of the Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, JKAW '94*, pages 65–71, 1994.

[53] M. Stefik, J. Aikins, R. Balzer, J. Benoit, L. Birnhaum, F. Hayes-Roth, and E. Sacerdoti. The organisation of expert systems, a tutorial. *Artificial Intelligence*, 18:135–127, 1982.

[54] D.M. Steier. CYPRESS-SOAR: A case study in search and learning in algorithm design. In P.S. Rosenbloom, J.E. Laird, and A. Newell, editors, *The SOAR Papers*, volume 1, pages 533–536. MIT Press, 1993.

[55] D.S.W. Tansley and C.C. Hayball. *Knowledge-Based Systems Analysis and Design.* Prentice-Hall, 1993.

[56] A.H. Vera and H.A. Simon. Situated action: A symbolic interpretation. *Cognitive Science*, 17:7–48, 1993.

[57] V.L V.L. Yu, L.M. Fagan, S.M. Wraith, W.J. Clancey, A.C. Scott, J.F. Hanigan, R.L. Blum, B.G. Buchanan, and S.N. Cohen. Antimicrobial selection by a computer: a blinded evaluation by infectious disease experts. *Journal of American Medical Association*, 242:1279–1282, 1979.

[58] S.M. Weiss, C.A. Kulikowski, and S. Amarel. A model-based method for computer-aided medical decision-making. *Artificial Intelligence*, 11, 145-172 1978.

[59] B.J. Wielinga, A.T. Schreiber, and J.A. Breuker. KADS: a modeling approach to knowledge engineering. *Knowledge Acquisition*, 4:1–162, 1 1992.

[60] G. Yost. Implementing the Sisyphus-93 task using SOAR/TAQL. In B.R. Gaines and M. Musen, editors, *Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 46.1–46.22, 1994.

[61] G.R. Yost and A. Newell. A problem space approach to expert system specification. In *IJCAI '89*, pages 621–627, 1989.

[62] Z. Zdrahal and E. Motta. An in-depth analysis of propose & revise problem solving methods. In R. Mizoguchi, H. Motoda, J. Boose, B. Gaines, and P. Compton, editors, *Proceedings of the Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW '94*, 1994.

Knowledge-level models of expert reasoning represent an important output of the knowledge- acquisition process, since they describe, in a conceptual and implementation-independent fashion, the different roles and types of knowledge required for a problem-solving task.Â Based on a knowledge-level description of ONCOCIN and on a review of the features of KADS, we developed K-ONCOCIN, a model of the ONCOCIN cancer-chemotherapy task that can be understood in terms of the four layers required by the KADS approach.In this paper, we offer a detailed description of the elements and layers of the K-ONCOCIN conceptual model. We also describe briefly a design model that implements K-ONCOCIN as a functional expert system. Knowledge modeling is a process of creating a computer interpretable model of knowledge or standard specifications about a kind of process and/or about a kind of facility or product. The resulting knowledge model can only be computer interpretable when it is expressed in some knowledge representation language or data structure that enables the knowledge to be interpreted by software and to be stored in a database or data exchange file. Knowledge-based engineering or knowledge-aided design is a process