Responses to "Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples" by B. Chandrasekaran

COMMENTS ON THE GENERIC TASK APPROACH

by Paul E. Johnson and Imran A. Zualkernan, *University of Minnesota*

Professor Chandrasekaran's paper shows important progress in our understanding of tasks requiring strong methods for solution. An important implication of Professor Chandrasekaran's work is the explicit separation of computational theory from a theory of algorithms and data structures (Marr, 1982). To some extent, such a separation already exists in the "weak methods" currently investigated in AI research. For example, it is possible to characterize means-ends analysis without prior commitments to a computational architecture. Unfortunately, when it comes to knowledge-based systems, each program is typically considered as an isolated experiment (Newell and Simon, 1976). The work of Chandrasekaran and others (e.g. Clancey, 1985; Johnson and Keravnou, 1985) represents an important direction in an attempt to understand the computational theories underlying these programs.

The key to Professor Chandrasekaran's approach is his statement that current descriptions of the knowledge used to develop expert systems are at the wrong level (too detailed), and his proposal that the level of rules be replaced by a level of domain problem solving objects (e.g. hypotheses). Arguments for the cognitive status of rules notwithstanding (e.g. Newell and Simon, 1972; Anderson, 1983), there can be little argument with the proposition that experts in a domain do not typically think in terms of rules. Clearly rules are a device of the knowledge engineer (or the cognitive scientist), to represent the knowledge required to perform tasks. What is at issue, is how to choose the appropriate level of domain knowledge and how to describe which such knowledge is needed to perform one of Professor Chandrasekaran's generic tasks. There does not yet appear to be a means of mapping domain knowledge onto a generic task, nor means for deciding what organization or structure among a set of generic tasks is most suitable for a specific problem solving content.

Professor Chandrasekaran suggests that his approach facilitates knowledge acquisition. Knowledge acquisition is a model building activity (McDermott, 1988) in which a knowledge engineer interacts with an expert to develop a problem solving model. This model plays an important role in system development. It tells the knowledge engineer what data to observe (what to extract from the domain or the expert) and what to ignore. As in other model building activity, one must have methods for data acquisition and model verification. To be an effective tool for expert system development, Professor Chandrasekaran's work needs to be augmented with a methodology for knowledge acquisition.

The goal of the knowledge engineering enterprise is to create a model of the knowledge required to perform tasks. Presumably such a model specifies the appropriate level of abstraction of domain knowledge as well as what kinds of task descriptions are appropriate (Johnson, Zualkernan and Garber, 1987). What is needed in order to test Professor Chandrasekaran's very plausible analysis of tasks is a comparable analysis of the expert side of the expert-task equation. Such an analysis would enable a specification of the knowledge to be embodied in an expert system and afford the kind of disciplined system evolution that is implied by Professor Chandrasekaran's approach.

References

Anderson, J R, 1983. *The architecture of cognition* Cambridge, MA: Harvard University Press.
Clancey, W J, 1985. "Heuristic classification" *Artificial Intelligence* **27**(3), pp. 289–350.

Johnson, L and Keravnou, E T, 1985. *Expert systems technology: A guide* Dover, NH: Abacus Press.
Johnson, P E, Zualkernan, I A and Garber, S, 1987. "Specification of expertise" *International Journal of Man-Machine Studies* **26**, pp. 161–181.
Marr, D, 1982. *Vision* pp. 19–31, San Francisco, CA: W H Freeman and Company.
McDermott, J 1988. "Exploiting task structure to automate knowledge acquisition" (invited talk). *The Seventh National Conference on Artificial Intelligence* August 21–26, St. Paul, Minnesota.
Newell, A and Simon, H A, 1976. "Computer science as empirical inquiry: Symbols and search" *Communication of the Association for Computing Machinery* **19**, 113–126.
Newell, A and Simon, H A, 1972. *Human problem solving* Englewood Cliffs, NJ: Prentice-Hall.

FUNCTIONAL ARCHITECTURES AND THE GENERIC TASK APPROACH

By Jean-Marc David, *Laboratoires de Marcoussis, CGE Research Center, Route de Nozay, 91460—Marcoussis, France*

Solving a non-trivial problem can rarely be reduced to performing a single, simple task. Within any knowledge-based system, there exists a collection of tasks to be performed in a cooperative fashion to solve the problem. Making these tasks explicit has been a recurrent concern over the past few years.

Prior to discussing the *generic task approach* proposed by Chandrasekaran, we will briefly review the different motivations for exhibiting a task level. Then, we will introduce the resulting *functional architecture* paradigm for designing knowledge-based systems, and replace the generic task approach in this framework.

1 Motivations for declarative task description

Knowledge-based systems perform a number of different tasks to solve a problem. Let us summarize the motivations for making these tasks explicit:

- *Describing problem-solving at a higher level.* There is a common feeling that, in order to describe what our systems are doing, we need a more relevant level of description than application-oriented descriptions or implementation-oriented descriptions. Without such a higher level, attempts to find a mapping between problems and knowledge-based architectures will fall short. The description of problem solving in terms of tasks provides a much higher level of description (Bennett and Engelmore, 1979; Clancey, 1984; Wielinga and Breuker, 1986).
- *Structuring knowledge acquisition.* Describing its cognitive activity at the task level is not just useful for a better understanding of the expert reasoning, but may also be used to guide knowledge acquisition. Roget (Bennett, 1984) and KADS (Breuker and Wielinga, 1985) are two systems that attempt to use such descriptions to perform semi-automated knowledge acquisition.

  Note that if the task structure is made explicit for knowledge acquisition purposes, this description is not necessarily part of the resulting application.
- *Improving strategic explanations.* Clancey's work on Neomycin (Clancey and Letsinger, 1981; Hasling, Clancey and Rennels, 1984) is probably the most famous example of the benefit that can be drawn from declarative task description to elaborate strategic explanations.

  Because the strategy is designed so that tasks make sense as things that experts actually do (e.g. "establish hypothesis space", "group and differentiate", "test hypothesis"), the representation of Neomycin problem solving activity at the task level provides the basis for sophisticated explanations.
- *Improving maintainability of large systems.* Soloway et al. have recently discussed the issue of coping with maintainability of R1/XCON rule base (Soloway, Bachant and Jensen, 1987). In particular, they have discussed how a task structuration should make debugging and modifications easier, by enhancing the readability and intelligibility of rules.
- *Designing "hybrid" systems.* Designing hybrid systems that integrate different knowledge representation models, different knowledge sources, and eventually different pieces of code, requires us

to precisely define the role played by each of these parts (Smith et al., 1984). This is especially the case when trying to make different reasoning paradigm cooperate (e.g. "deep" and "shallow" reasoning (Simmons and Davis, 1987; Steels, 1988).

These are among the different motivations that have led to the design of knowledge-based systems directly at the task level.

2 Functional architectures

Knowledge-based systems designed within *functional architectures* are built from a collection of explicit tasks, where:

1 Each task plays a precise *role* in the problem solving, and thus has precise semantics.
2 The role of each task can be defined by an information processing function that relates its output to the information given at the input.
3 Each task has its own knowledge, in its own formalism, to achieve its function (this is what distinguishes functional design of knowledge-based systems from traditional functional programming), and its own strategy.
4 Tasks can be decomposed into subtasks; this decomposition is pursued till tasks can no longer be decomposed, but need to be performed by applying knowledge. The strategy of a task can be defined as the way this task controls its subtasks, or controls the application of its knowledge.

Systems designed within the generic task framework obviously meet these criteria. Other examples of this approach, not based on generic tasks, are Crystal (Smith et al., 1984), an interactive log-interpretation system, and DIVA (David and Krivine, 1988), an expert system for vibration-based monitoring of large rotating machinery.

The aim of DIVA is to help a plant operator to interpret vibration evolutions to diagnose developing faults. At a higher level, DIVA tries to identify the problem situation before elaborating its diagnosis. Both tasks (*situation recognition* and *diagnosis*) require information about the problem that is given by a third task: the *data abstraction and information retrieval* task.

The role of the situation recognition task is to characterize, as precisely as possible, the observed problem; roughly speaking, it is achieved through the use of prototypical situation descriptions, in an almost similar way to Centaur (Aikins, 1983).

Advantages of designing DIVA within a funtional architecture are in respect with the motivations previously quoted:

- The description of what the system is doing is more accurate. Each of the three tasks mentioned above can be further decomposed into its subtasks; DIVA can thus be described from a collection of around 20 specific tasks and subtasks. The strategy—how tasks are related to each other—can also be easily described.
- Knowledge to be acquired is precisely identified through its role in the problem solving. Moreover, the architecture provides a framework that makes clear what has already been acquired, what is missing, and the nature of missing knowledge.
- Explanations are based on the task structure, and the semantics of tasks. Currently, the explanation module matches some explanation patterns onto the trace of tasks and aggregates them to produce strategic explanations with various levels of details. But, whatever the chosen explanation paradigm, the representation of problem solving behaviour at the task level provides a rich basis for explanation.
- Structuring a large knowledge base through a collection of tasks and subtasks has proved to facilitate debugging, validation and maintainability. This is not surprising, and has to be linked to more traditional software engineering practices.

So, this real world experiment has shown, if necessary, that designing knowledge-based systems within functional architectures presents a significant contribution.

3 The generic task approach

Let us now come back to the generic task approach. This approach, though prior to most of the systems mentioned above, can be seen as a further step in designing knowledge-based systems within functional architectures. Its two main contributions are that it provides a theory to analyse problem solving activities, as well as a set of tools directly at the task level.

3.a Generic task as a problem solving theory

The representation of a knowledge-based system within a functional architecture does not impose constraints on the task structuration; the only requirement is that the identified tasks should have strong enough semantics, e.g. to support explanatory purposes. Consequently, there is a large number of different possible structurations.

The limits are thus clear. There is no guide in the analysis of problem solving, since any particular structuration is eligible. Likewise, there is no, or few, possibilities to compare systems, and understanding what a system does may require the complete analysis of the architecture (since no assumptions can be made about high-level tasks).

On the contrary, the identification of a set of generic tasks provides primitive concepts to describe problem solving activities. So, it provides a guide in the analysis of any problem solving activity; and because generic tasks can act as a common and shared vocabulary, systems can be defined without any ambiguity (this has to be compared with the above reference "*in an almost similar way to Centaur*").

3.b The generic task toolset

The lack of generic tasks implies a lot of rewriting: tasks, explanation modules (or at least some patterns), redefining knowledge acquisition methods, and so on. *A contrario*, from the identification of some tasks as generic, it becomes worthwhile to design associated tools.

From a pragmatic (or engineering) point of view, these tools will be used as higher-level building blocks. This does not mean that these building blocks are sufficient, since the theory is surely not yet complete. But, while functional design of knowledge-based systems was previously done with traditional (low-level) programming languages, they bring closer description and implementation.

From the generic task theory point of view, these tools are necessary to perform experiments, in order to validate and refine the already identified tasks, the way they cooperate and can be controlled. They also prepare the ground to study new problem solving areas, to complete the theory.

4 Conclusions

Designing knowledge-based systems within functional architectures has proved to be a powerful knowledge engineering methodology. This approach obviously does not claim to solve all problems, but nonetheless presents a significant contribution, as proved by some real world applications.

The generic task approach can be seen as a further step in this framework, by providing a theory of problem solving, and a set of tools to implement it. How generic are the tasks already identified, and consequently the shells provided in the generic task toolset, is now a matter of empirical study. The point is that, thanks to the work carried out by Chandrasekaran et al., the discussion has now shifted to a higher level.

Acknowledgement

I am indebted to Jean-Paul Krivine for the work carried out on the DIVA project, and for many discussions on the ideas expressed here.

References

Aikins, J S, 1983. "Prototypical knowledge for expert systems" *Artificial Intelligence* **20**(2), pp. 163–210.

Bennett, J S and Engelmore, R S, 1979. "SACON: a knowledge-based consultant for structural analysis" *Proc. of the Sixth IJCAI*, Tokyo, Japan, August.

Bennett, J S, 1984. "ROGET: Acquiring the conceptual structure of a diagnostic expert system" *Proc. of the IEEE Workshop on Principles of Knowledge-Based Systems*, Denver, Colorado, December.

Breuker, J A and Wielinga, B J, 1985. "KADS: Structured knowledge acquisition for expert systems" *Proc. of the Fifth International Workshop on Expert Systems and their Applications*, Avignon, France, May.

Clancey, W J and Letsinger, R, 1981. "NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching" *Proc. of the 7th International Joint Conference on Artificial Intelligence*, Vancouver, Canada.

Clancey, W J, 1983. "The epistemology of a rule-based expert system—a framework for explanation" *Artificial Intelligence* **20**, pp. 215–251.

Clancey, W J, 1984. "Classification problem solving" *Proc. of the AAAI National Conference on Artificial Intelligence*, Austin, TX, August.

David, J-M and Krivine, J-P, 1988. "DIVA: an expert system for vibration-based monitoring of large rotating machinery" (in French). Final report, Laboratoires de Marcoussis, May.

Hasling, D W, Clancey, W J and Rennels, G, 1984. "Strategic explanations for a diagnostic consultation system" *Int. J. Man-Machine Studies* **20** pp. 3–19.

Simmons, R G and Davis R, 1987. "Generate, test and debug: Combining associational rules and causal models" *Proc. of the Tenth IJCAI*, Milan, Italy, August.

Smith, R G, Lafue, G M, Schoen, E and Vestal, S C, 1984. "Declarative task description as a user interface structuring mechanism" *Computer* **17**(9) pp. 29–38, September.

Soloway, E, Bachant, J and Jensen, K, 1987. "Assessing the maintainability of XCON-in-RIME: Coping with the problems of a VERY large rule-base" *Proc. of the AAAI National Conference on AI*, Seattle, Washington, July, pp. 13–17.

Steels, L, 1988. "Components of expertise" *AI Memo no 88–16*, Vrije Universiteit Brussels; July.

Wielinga, B J and Breuker, J A, 1986. "Models of expertise" *Proc. of the ECAI 86 Conference*, Brighton, UK.

GENERIC TASKS OR WIDE-RANGING KNOWLEDGE BASES? by

Yumi Iwasaki, Richard Keller, Ed Feigenbaum

Abstract

The need for a large, multi-use knowledge base is discussed in the light of Chandrasekaran's argument for generic task toolbox.

In *Generic Tasks as Building Blocks for Knowledge-Based Systems: The Diagnosis and Routine Design Examples*, B. Chandrasekaran argues that the current generation of knowledge representation languages such as rules, frames, or logic are too low-level to be useful tools for building problem solving systems. He writes that there is a need to build a set of higher-level language constructs, each of which is appropriate for a particular problem solving strategy. He goes on to describe the set of such strategy-specific tools he and his colleagues have built.

We agree with B. Chandrasekaran's argument for the need to build task-generic tools. However, we feel that he fails to address an important issue which must be addressed if his generic task toolbox is to be more than simply a bag of tools without a common medium to unify them into an integrated system. The issue is that of knowledge and its representation. His overall approach consists of identifying generic tasks and building a separate tool embodying the problem solving strategy useful for carrying out the generic task. This approach encourages construction of many isolated knowledge bases using knowledge representation forms which are also very specific to particular tasks. We feel that such an approach leads to much wasted effort in constructing a large number of knowledge bases whose knowledge may be encoded in different forms but may arise from the same body of fundamental knowledge of the domain.

One of the major shortcomings of the first-generation expert systems is the difficulty of transferring or reusing knowledge. The knowledge encoded in the first-generation expert system rules is very specific to a narrow domain of expertise and is encoded in a form that is sufficient for carrying out a single problem solving task (e.g. diagnosis). Therefore, these systems only need a relatively

simple control structure. Chandrasekaran's tools also utilize this type of "shallow" knowledge, although not necessarily in the form of rules. The simplicity of this approach facilitates rapid construction of problem solving systems that achieve a high level of performance, makes possible the building of problem solving shells, and enables problems to be solved efficiently. However, the task-specificity of both the form and the content of the knowledge in these systems makes it very difficult, if not impossible, to use the knowledge for any other task in even the same domain.

It takes much effort to build a knowledge base dealing with practical problems in even a moderately complex domain. If, after building a large knowledge base for a particular type of task—say design—in a certain domain, one has to build an entirely new knowledge base from scratch for another task—say diagnosis—in the same domain, then a large part of the effort to build the second knowledge base would seem redundant; the new knowledge base will consist of largely the same body of knowledge, only cast in a different task-specific form.

Possibly, then, a better alternative is to build one general-purpose knowledge base. Such a unified-knowledge base approach will be useful in many domains where there are several tasks each requiring different problem solving strategies, and knowledge which may be in a task-specific form, but which nonetheless must be consistent across tasks. For example, consider the process of producing automobiles. Instead of building a separate knowledge base to support each of the tasks in development and manufacturing of slightly different types of automobiles, such as design, planning, process control, testing, diagnosing, etc., it might be better to build a large knowledge base containing common knowledge pertaining to all automobiles. Such knowledge includes knowledge of the physical structures, physical principles, commonsense knowledge regarding cars, government regulations, empirical knowledge, etc.

Such a general knowledge base can be directly interpreted by programs to carry out different types of tasks at the level of so called "deep reasoning" or "first principles". Alternatively, task-specific knowledge bases such as those advocated by Chandrasekaran can be generated from the general knowledge base, and task-specific inference engines can access these knowledge bases to achieve a greater degree of efficiency.

At Knowledge Systems Laboratory, we are beginning a project that has as one of its goals construction of a large, multi-use knowledge base. We are studying issues involved in making it as general and multi-functional as possible. We acknowledge that it may not be possible in general to represent knowledge without explicit or implicit assumptions about the use of the knowledge represented. However, the representation itself should provide a means to make such assumptions as explicit as possible. Also, the representation must offer a rich enough vocabulary to represent a wide range of general to task-specific knowledge and assumptions underlying each part of the knowledge.

We aim to build this knowledge base to support multiple tasks and multiple domains in physics and engineering. There is a large body of fundamental physics and engineering knowledge that is shared across many domains of applicability. For example, knowledge of basic thermodynamics is relevant to problem solving in a large number of device domains, such as refrigerators, air conditioning systems, and power plants, to name only a few. Domain-specific, "shallow" knowledge of the kind that Chandrasekaran's tools rely on can be considered special-purpose forms of such general knowledge.

We are also studying the issue of knowledge transformation. Once we have a general-purpose large knowledge base, we can refine the general knowledge by augmenting it with more information about a task and a particular domain. In this manner, we hope to construct layers of more and more specialized knowledge. Conversely, specialized knowledge can be justified in terms of more general knowledge for the purpose of providing explanations.

Without efforts to represent the large body of common knowledge underlying task-specific knowledge bases, the approach of building strategy-specific problem solving tools will lead to proliferation of task-specific knowledge bases that have no relationship to one another even if they deal with the same domain. If we knowledge engineers want to avoid the fate of having to build many knowledge bases that are different in form but essentially similar in their content, we must start trying to capture that underlying general knowledge in a large, unified knowledge base.