THE GRADUATE COURSE ADVISOR: A MULTI-PHASE RULE-BASED EXPERT SYSTEM

Marco G. Valtorta *

Bruce T. Smith **

Donald W. Loveland *

ABSTRACT

The Graduate Course Advisor (GCA) is a rule-based expert system that advises graduate students in Computer Science. It is implemented in Prolog, using an inference engine modeled after MYCIN's. The advising task is divided into four phases, each of which may apply the inference engine to its own rule base and invoke other procedures. Some phases are diagnostic (e.g., determining a student's needs), whereas others involve planning (e.g., planning a student's schedule). The decomposition helps to manage the complexity of the advising task, it simplifies the knowledge engineering problem, and it will enhance the clarity of explanations. However, the decomposition is tightly bound to basic design decisions; a major decoupling in the GCA followed from our assumption that best schedules are almost always chosen from the top-valued courses.

1. Application Domain and History of the GCA

The Graduate Course Adviser (GCA) is a rule-based expert system that simulates a faculty adviser in suggesting the schedule of courses a student should take. It knows about all courses offered by the Department of Computer Science at Duke University, as well as courses frequently taken by Computer Science students but offered by other departments. It also knows about departmental and university regulations concerning graduate degree programs. The GCA obtains information about a student's academic history and interests during an advising session. (Table 1 shows some questions usually asked by the GCA.) The GCA was designed to be easily modified by the Director of Graduate Studies, without help from the designer or maintainer.

Table 1. Typical questions from the GCA's phases.

schedule length phase	Have you passed your prelims yet? Do you enjoy full support through a fellowship or a scholarship?
legal course phase	Is this your first semester in the Computer Science program at Duke? Please enter the names of the computer science courses you have taken, as a list.
course evaluation phase	Have you ever worked as a programmer? Which computer language are you most familiar with? Do you like computer games? Do you like courses with an emphasis on lab work?

Marco Valtorta designed and implemented the original version of the GCA as his Master's project, under the supervision of Prof. Donald Loveland. The GCA uses a rule interpreter ("inference engine") based on one described by Hammond [Hammond, 1982; Clark and McGabe, 1980], which is a Prolog implementation of the MYCIN inference engine [Shortliffe, 1976]. The original version of the GCA was written in Edinburgh Version 7 Prolog and ran on a PDP-11/70 [Clocksin and Mellish, 1981]. Because of the limitations of Version 7 Prolog and the PDP-11/70, the GCA was ported to a VAX 11/780 at the University of North Carolina at Chapel Hill (UNC) and C-Prolog [Pereira, 1983]. At the same time, the

* Department of Computer Science, Duke University, Durham, N.C. 27706. M.G. Valtorta and D.W. Loveland have been supported in part by the Air Force Office of Scientific Research, under grants AFOSR-81-0221 and AFOSR-83-0205.

** Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, N.C. 27514. B.T. Smith has been supported in part by ARO grant number DAAG29-83-K-0090.

CH2104-8/84/0000/0053$01.00 © 1984 IEEE

Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems, 1984, ISBN 0-8186-0567-7

GCA was reorganized into a modular program whose structure better reflects the application domain. Bruce Smith has participated in the development of the VAX version, at UNC. Several Duke students have also contributed to this project.

The GCA's rule bases now contain nearly 400 rules. The system is in its testing stage and has advised students on an experimental basis for two semesters [Valtorta, 1984].

2. Multi-Phase Architecture

Advising a graduate student is a four-phase process:

(a) determine how many courses the student should take;

(b) determine which courses the student may take, based on prerequisite requirements and the student's academic history;

(c) determine the "best courses" for the student;

(d) generate potential schedules from the best courses and consider the interactions among the courses to determine the "best schedule" of the appropriate length.

These four phases are implemented in the GCA as four expert system modules, sharing the same rule interpreter but using different rule bases. Phases (a) and (b) determine the needs of the student, and are therefore of a *diagnostic* character, while the last two phases *plan* the student's schedule [Stefik et al, 1982].

A *manager* program directs the activity of the modules. Communication between them is through facts asserted to the Prolog data base. Each phase consists of a *driver* that may set up one or more goals for the rule interpreter or call some "algorithmic" procedures. In the current version of the GCA the first, third and fourth phases depend primarily upon the rule interpreter, while the second phase uses the rule interpreter only minimally. (See Figure 1.)

Facts in the data base have a uniform representation as MYCIN-like *object-attribute-value* triples. (Table 2 shows three triples as recorded in a GCA session log. The third element of each triple, labeled "Hypotheses", is a list of *value-CF* pairs. The GCA's certainty factors, or *CF*s, range in value from -1000 to 1000.) Each phase has access to the facts asserted by previous phases, and it adds its conclusions to the data base for use by the following phases. Communication is simple, because the phase decomposition mirrors the natural structure of the problem.

The rules in the four phases' rule bases have a common structure. They use the same rule interpreter, which does not distinguish among rules from different rule bases. Moreover, the phases share the tools that make up the user interface. This is possible because the driver programs and rule bases of the four phases have no knowledge of the details of the data structures or the user

Table 2. Typical assertions to the Prolog database, as recorded in the session log.

Asserting answer from user		
Object	:	student
Attribute	:	employed
Hypotheses	:	[v(yes,-1000)]
Asserting answer from user		
Object	:	student
Attribute	:	reads_JACM
Hypotheses	:	[v(yes,900)]
Asserting deduction		
Object	:	student
Attribute	:	is_interested_in_theory
Hypotheses	:	[v(yes,990)]

interface, an example of the "information hiding" technique advocated by Parnas [Parnas, 1972]. This allows a common format for storing and accessing facts in the database, asking questions, recovering from input errors and keeping a session log. Since all the questions are in the same format, users do not realize that questions come from different expert rule bases.

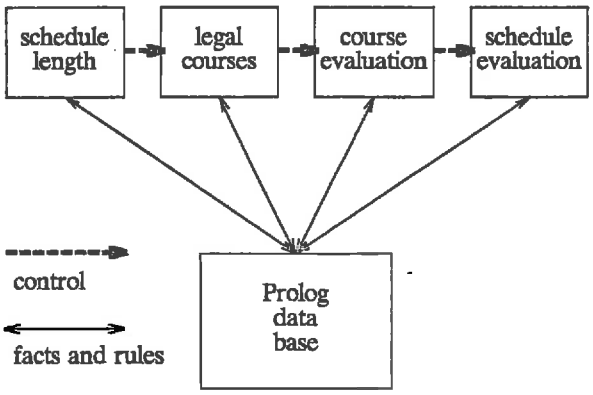

Figure 1: Interaction among the GCA's experts.

3. Advantages of a Multi-Phase Architecture

The original GCA contained only two of the four phases: a course evaluation phase and a schedule evaluation phase. The first of these phases evaluates elements, while the second evaluates sets of elements. This kind of decomposition would be useful in any domain with a "weak interaction" between the evaluation of elements and the evaluation sets of

elements. That is, it is chiefly the values of its elements that determine a set's value. The elements' interactions are only secondary. We emphasize that this decoupling is directly tied to a model of our problem, i.e., that best or near-best schedules are subsets of the set of highly-ranked courses.

We note that our model is not perfect. There are situations where a student would choose a schedule containing a course of only modest value (by the student's own assessment) course simply "because it is different" or "for a change of pace". Our assumption of weak interaction among courses in a schedule is clearly invalid here, but we note that cases where a student goes quite far down the list of preferred courses are rare. And in those cases even a human advisor is a poor predictor. We console ourselves with the view that the GCA is a "rational advisor", suggesting courses to take if the student were not emotional in his selection.

Examples of other domains where this model seems appropriate include inventory determination and portfolio analysis. For example, to determine a good inventory for a sports shop the owner would consider the items which might be carried, then he would consider how they fit together. Suppose that soccer balls and golf balls both had a market. The shop owner might decide to carry only one kind of soccer ball and one kind of golf ball, rather than two kinds of soccer ball and no golf balls. He would be able to choose the most profitable kind for each by considering soccer balls and golf balls separately, even if either soccer ball were somewhat more profitable than the golf balls.

During the development of the GCA, two more phases were added: the schedule length determination phase and the legal course determination phase. (A course is *legal* if a student may take it. That is, he has not completed the course previously, and he has completed all its prerequisites.) Schedule length rules were extracted from interviews with the Director of Graduate Studies (DGS). The current DGS has a strict policy concerning schedule length that involves only a few factors, such as as type of support. This made the knowledge engineering process for this phase fairly simple.

Determination of legal courses currently follows a literal interpretation of the prerequisites for a course:

- a student may take a course if he has completed its prerequisites, and
- a student who has taken a course has completed its prerequisites.

Legal course determination was originally implemented by rules. Under the literal interpretation of "legal course", the phase proved simple enough to be rewritten as an "algorithmic" procedure, along with a few rules to determine whether a student should take an introductory data structures course. The more general problem of legal courses includes determining whether a student has completed the equivalent of Duke courses elsewhere. We anticipate that this phase, when complete, will be an expert system as complex as the phase that determines the best courses. Mr. Tim Harrison is working on this problem, with Ms. Corinna Van Der Veen and the authors. The goal is a rule base of course-equivalence rules.

The reader might wonder why the schedule length phase was not also rewritten as an algorithm, since it is rather straightforward. There are two reasons. The first is to anticipate change: different Directors of Graduate Studies often use different policies in the determination of schedule length. (Duke is also considering a basic policy change regarding courses per semester.) The second is that the rules mirror the guidelines the DGS communicated during interviews. A failure of this phase would probably indicate a case the expert had overlooked, and this facilitates debugging. In general, we implemented each phase as a rule-based system the first time.

The four phases correspond to the way our experts view the advising task. This has simplified the knowledge engineering process, because the experts can be queried on each phase separately. We have been able to debug each rule base independently, greatly reducing the number of rule interactions. Students who have used the GCA found its questions ordered into the four natural areas. (E.g., in the first phase, the GCA asks about financial support and degree program. In the second, it asks about academic histories.) The decomposition has also helped new members of the GCA project understand the system.

The decomposition into four phases has limited the size of the individual rule bases, helping to manage the complexity of the GCA program. For example, postponing consideration of the interactions among courses until the last phase allows the GCA to discard obviously unsuitable courses during the third phase. This limits the number of schedules to be generated and evaluated during the final phase.

4. Comparison of the GCA with Other Expert Systems

MYCIN uses context trees for two purposes. "The context tree is useful not only because it lends structure to the clinical problem, but also because during the consultation it is often necessary to identify relationships between two or more different contexts" [Narain, 1981]. The phases in the GCA play a role similar to context trees, but there are differences. First of all, each phase need not be implemented as a rule-based system. (One phase is primarily algorithmic, legal course determination, as explained earlier. Only the schedule length phase sets up a single top level goal, MYCIN-style.) Secondly, MYCIN's chains of rules may span several contexts. For example, rules dealing with an organism can be invoked to solve a goal set up by rules dealing with a culture. In the GCA, rules for determining a value for any particular

attribute only occur during a single phase. (A hypothetical MYCIN-like context tree for determining the best schedules is shown in Figure 2. It should be compared with the description of the GCA in Figure 1.)

The fact that the GCA's phases need not be entirely rule-based helped us avoid a major problem in MYCIN's limited control structures. As Waterman and Hayes-Roth point out [Waterman & Hayes-Roth, 1983], the lack of support for iteration in MYCIN's language makes it difficult to write simple rules for tree searching. In the GCA, the corresponding problem is generation of schedules, as combinations from the list of best courses. This is done in the schedule evaluation phase by having the driver first generate combinations of courses, then calling a top level rule to evaluate each schedule.

The restriction of an inference tree to a single phase in the GCA has several consequences. This localization of knowledge simplifies the tasks of debugging and knowledge engineering. It should also simplify explanations, when that facility is added to the GCA. It contributes to the comprehensibility of the GCA, since one can learn about one phase at a time. A negative aspect is the lack of contextual information when evaluating schedules, but the weak interaction between elements and sets in our domain minimizes this effect.

A certain type of decomposition can be said to occur in frame-type knowledge-based systems such as CENTAUR. There, the prototypes represent diagnoses, which can be viewed as abstractions of clusters of patient data points. Hypothesizing a particular prototype allows one to focus attention temporarily on a small neighborhood of the problem space and probe "within context". This temporary localization is a type of decomposition where context temporarily reduces the degrees of freedom. (Or, in AI terms, solves temporarily the frame problem.) Indeed, "separate sets of expertise are applied during different stages of processing in CENTAUR by grouping the production rules according to their functions and applying them at different points in the consultation" [Aikins, 1983, p.201]. Both CENTAUR's prototypes and the GCA's phases eliminate many of the shortcomings of "monolithic" rule-based systems described by Aikins. A major difference between the GCA and CENTAUR is in the systems' control mechanisms. In the GCA there is a simple, serial flow of control through the phases, while CENTAUR uses complex agendas.

The GCA project resembles the CENTAUR project in another way. CENTAUR is a "second-generation" system that performs the same function as an earlier MYCIN-like system called PUFF. The original GCA was a more MYCIN-like rule-based system. It has been re-written to emphasize the division into phases and to exploit information hiding in the user interface and data representations.

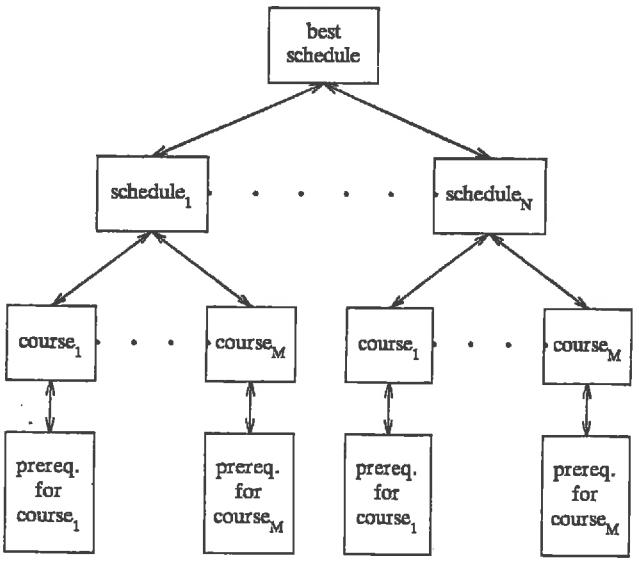

Figure 2: A MYCIN-like context tree for the GCA's domain.

5. Conclusion

The advantages of the multi-phase architecture to the GCA are summarized as follows:

(a) simplified interactions with experts,

(b) effective modularization,

(c) reduction of complexity, and

(d) possibility of substituting algorithmic or "mixed" phases for purely rule-based phases.

A good modularization has many advantages in the development of large software systems [Parnas, 1972; Parnas et al, 1984]. Parnas identifies them as ease of management, product flexibility, and comprehensibility. We have enjoyed all these benefits in the GCA's development. We have also enjoyed the benefits of information hiding, namely changeability (demonstrated by the rewriting of the prerequisite phase), independent development (taking place in the work on course equivalence) and comprehensibility.

Since the GCA is not a mature system, we cannot draw definite conclusions about criteria for decomposition of expert systems. It is paramount to keep the knowledge engineering process in mind when dividing expertise into modules. Specifically, it should be possible to interview the expert on just one of the phases, without his worrying about the other phases. The success of the GCA's architecture is largely due to the fact that the domain expert, the DGS, suggested its subdivision into phases.

The use of decoupled processes brings with it the concern that global direction, i.e., "focus of attention", could be lost. The GCA seems not to suffer from a lack

of global control, and the reason is the obvious one: a problem decomposes well when a globally good solution is the product of locally good solutions. For example, the modules within the GCA have simple and sound termination criteria, so no module remains in control when an overseer might have removed control.

We think that multi-phase architectures will prove equally effective in solving other problems involving the sequential application of several kinds of expertise. Somalvico, Colombetti, Guida, and Meltzer have very recently proposed a similar architecture for an expert system for personal financial planning [Somalvico, 1984].

BIBLIOGRAPHY

Aikins, J.S. "Prototypical Knowledge for Expert Systems," *Artificial Intelligence*, 20 (1983), 163-210.

Clark, K.L. and F.G. McGabe. "Prolog: A Language for Implementing Expert Systems." Typescript, Department of Computing, Imperial College, London, England, November 1980.

Clocksin, W.F. and C.S. Mellish. *Programming in Prolog.* New York: Springer-Verlag, 1981.

Hammond, P. "Logic Programming for Expert Systems." Technical Report: Doc 82/4 (March 1982). Department of Computing, Imperial College, London.

Narain, S. "MYCIN: The Expert System and Its Implementation in LOGLISP." Technical Report 6-81, School of Computer and Information Science, Syracuse University.

Parnas, D.L. "On the Criteria to be Used in Decomposing Systems into Modules." *CACM*, 5, 12 (December 1972), 1053-1058.

Parnas, D.L., Clements, P.C. and Weiss, D.M. "The Modular Structure of Complex Systems." *Proceedings of the Seventh International Conference on Software Engineering.*

Pereira, F. "C-Prolog User's Manual, Version 1.2a", Department of Architecture, University of Edinburgh, March 1983.

Shortliffe, E.H. *Computer-Based Medical Consultation: MYCIN.* New York: American Elsevier Publishing Co., 1976.

Somalvico, M. "Nasce al Politecnico di Milano un Computer con gli Schemi dell'Intelligenza Artificiale." *Corriere Della Sera*, Milan, Italy, May 1, 1984, p. 15 (In Italian).

Stefik, M., J. Aikins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti. "The Organization of Expert Systems: A Tutorial." *Artificial Intelligence*, 18 (1982), 135-173. (Also in Hayes-Roth, Waterman, and Lenat, eds. *Building Expert Systems.* Reading, Massachusetts: Addison-Wesley Publishing Company, 1983.)

Waterman, D.A. and Hayes-Roth, F. "An Investigation of Tools for Building Expert Systems," in Hayes-Roth, Waterman and Lenat, eds. *Building Expert Systems.* Reading, Massachusetts: Addison-Wesley Publishing Company, 1983.

Valtorta, M. "Knowledge Refinement in Rule Bases for Expert Systems: An Application-Driven Approach." To be presented at the First International Workshop on Expert Database Systems.