In *Second Generation Expert Systems*, J. M. David, J. P. Krivine and R. Simmons, Eds. New York: Springer Verlag. 467-476, 1993. Also appeared in *IEEE Expert*, 6(5) 61-65, 1991.

Using the System-Model-Operator Metaphor for Knowledge Acquisition

William J. Clancey, Institute for Research on Learning
Monique Barbanson, Metaphor Computer Systems

The system-model-operator framework provides a unifying perspective for the ways that expert systems represent, organize, and apply knowledge. This lets us reuse domain-general knowledge and chart the knowledge-acquisition effort needed to extend task-specific shells to other domains.

The techniques for building expert systems have advanced from tools that provided an "empty knowledge base" with a backward-chaining inference engine, such as Emycin,[1] to tools that allow for an explicit representation of the domain-general control knowledge necessary for a specific task, such as diagnosis or design.[2 6] Task-specific tools incorporate a way of organizing knowledge and an inference procedure for applying this knowledge. As researchers analyze these tools to generalize and integrate different methodologies, we need to understand the relation between specific problems and general ways in which knowledge can be organized and applied. In essence, how should we formalize the reusable part of the knowledge base so its capabilities and limits can be related to new problems? Task-specific architectures need to address the following distinct issues:[7]

- What system is being modeled?
- For what purpose, or task, is the system being modeled?
- What subsystems and subprocesses are represented in the general model (that is, in the knowledge base)?
- What subsystems and subprocesses are represented in a situation-specific model (that is, in a problem solution)?
- What relational networks (what kinds of hierarchies and transitional graphs) are used to represent processes?
- What are the operators, or inference procedures, for constructing situation specific models?
- How are the relational networks and inference procedures implemented in a programming language (for example, in frame and rule-based languages)?

The system-model-operator metaphor. To introduce the idea of this metaphor, we begin with the idea of describing knowledge bases in terms of tasks, systems in the world, and situation-specific models, pre sented several years ago as heuristic classification.[3] We extend that idea to describe inference as the process of constructing and comparing situation-specific models of processes for some purpose. Situation specific models are usually chained: One system's model (for instance, a diagnostic model of physiological processes) feeds into decisions for constructing another system or process (for example, therapy). We can view each situation-specific model as a graph, and inference subprocedures as operators for manipulating the nodes and relations within this graph and across graphs representing other situation-specific models. Clancey explores this idea in detail and relates it to other control paradigms such as blackboards.[7]

We can illustrate this idea in the medical-diagnosis context of Heracles-DX, the diagnosis shell developed from Neomycin. Heracles uses subtasks and metarules to represent the inference procedure. Sub tasks are procedures that order and control the application of conditional statements. called metarules. Metarules are

stated in a language of relations representing the mod eled system's structures and processes (for example, causal and subtype relations). The general model of the domain is expressed as a set of propositions over these relations. Metarules use variables ratherthan domain terms (so we say they are domain general). A given set of propositions about a particular situation in the world constitutes a situation-specific model. The relations and operators are the reusable knowledge in Heracles-DX, constituting a language for representing general and situation-specific models, plus an inference procedure. According to the system-model-operator view, subtasks are operators for manipulating situation-specific graphs that represent the structures and processes in the system being modeled.

This perspective provides a unifying basis for analyzing expert systems by reducing the dimensions of domain, problem type, and problem-solving method to a graph representation. As a general architecture, Heracles lets us define subtasks. metarules, and relations, but does not contain any specific operators or relations. But can we use the Heracles architecture to efficiently write new subtasks and meta rules for a different, nondiagnostic task? We designed an experiment to find out: We developed Topo, an expert system for configuration, using the Heracles shell.

Blackboards and operators for computer network layout

Topo explores the problem of writing new subtasks and metarules for the logical topology design of computer networks. This problem is relevant to efforts to develop a front end for Digital Equipment Corporation's programs for computer system configuration. For example, an expert system like Topo might provide a sales person with a language for modeling a potential client's business and information-processing requirements. Topo is designed to propose a layout of generic components and connections, suitable as input to sizing and configuration programs such as Xcon.[9] The program we describe in this article is a prototype that runs on one example case. Our interest here is not in solving the logical-topology problem, but in determining how the system-model operator perspective and the availability of the subtask/metarule language help or hinder knowledge acquisition.

Designing Topo in terms of process models. Topo first constructs a model of the client's business and then produces a corresponding network design. The program reasons as follows:

(1) Construct a model of the physical and organizational structure of the client's business. Describe the business sites and floor locations of work groups at each site.
(2) Determine the information processing requirements, leading to a preliminary sizing (for instance, the number of printers required).
(3) Design the network topology:
- Derive the backbone from the physical layout.
- Represent the components and connections.

Following the lessons learned from heuristic classification, we first conjectured that Topo would contain three kinds of situation-specific models corresponding to these three steps. However, we found that a separate case-based reasoning system could estimate the type and number of devices more conveniently. Therefore, we did not require a separate model for information-processing requirements. In the final design, Topo's information-processing requirements were directly represented as properties of workgroups within the physical and organizational model rather than as two separate models chained together. In effect, elements of the physical and organizational structure (such as sites and buildings) are related, as are elements of the network topology (for instance, segments and backbones). Conceptually, the inference procedure for Topo defines subtasks that construct a situation-specific model for each connected system. Each subtask defines metarules, each of which characterizes a set of domain rules that can be applied to construct a piece of the situation-specific model, such as Business-at-site-1 or Network-at-site-2, shown in Figure 1. The two blackboards (business and network) contain a panel for each site (corresponding to a situation specific model), and the levels in each panel correspond to hierarchical connections (for instance, segments on a backbone).

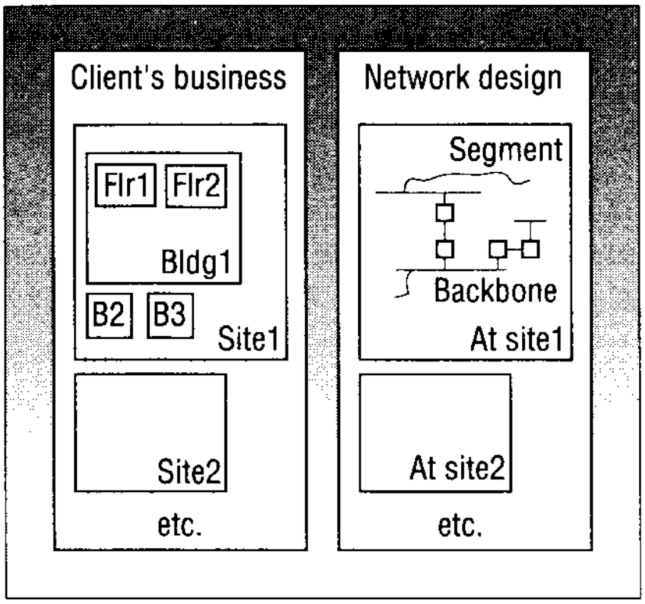

Figure 1. Two blackboards in TOPO: a client's business (left); network design (right).

Defining operators for constructing configuration models. Generalizing from the representations used in Topo, we can define configuration-specific inference operators as follows:

(1) Draw separate pictures for each real world system being modeled.
(2) Specify the class structure of nodes in each situation-specific model, and draw separate situation-specific models for unconnected systems (for example, different sites).
(3) Draw links showing the information mapping between models of different types (for example, from work group to equipment layout).
(4) Describe operators for placing nodes in each model, linking them, and specifying their spatial and process attributes.

In practice, we abstract descriptions of Topo's operators (see Figure 2) from the subtasks and metarules that carry out the necessary computations. A given operator typically corresponds to a single Heracles subtask (and hence several metarules). Higher-level subtasks control when the operators are performed (just as Process-hypothesis in Neomycin determines that the operator Test-hypothesis is executed before Refine-hypothesis). As shown in Figure 2, the six operators of Topo's inference procedure place different nodes and links in the situation-specific models. Another program computes the sizing table used by Op3 (quantifying the amount of equipment).

Based on the perspective of Clancey's analysis of blackboard systems,[7] in which he showed the correspondence between the blackboard paradigm and the system model-operator view, we can describe Topo's operators in more detail in terms of the nodes and relations they manipulate on the blackboards (see Table 1). The relations of the situation-specific model are specializations of the general domain model. For example, the general domain model can indicate a Uses/consumes relation between types of organizational structures and types of services. A client business situation-specific model indicates that a particular organizational structure uses or consumes a particular service (for example, "Work-group-3 Uses/consumes Information-storage"). Topo implements operators as Heracles subtasks; therefore the relations of Table 1 appear in metarule premises. The situation-specific model's nodes (terms of the relations) appear as variables in the metarules.

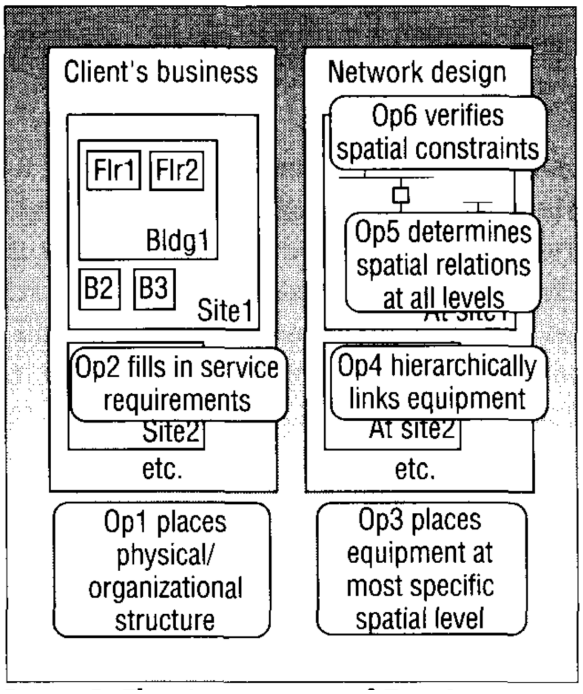

Figure 2. The six operators of TOPO's inference procedure.

Table 1. Domain relations used by TOPO's operators. POS is the physical/organizational structure, and sizing is the capacity of a service or supply.

Operator	(Relation Node1 Node2)
"Determine Position"	(Located-in/at POS <place>)
"Determine Service Requirements"	(Uses/consumes Organizational-Structure Service)
"Process Sizing Data"	(Used-by Organizational-Structure Equipment) (Provides Equipment Service) (Amount-Supply Service Sizing-data-table) (Sizing.Data.Table Equipment #)
"Derive Logical Topology from POS and Equipment "	(Provide-Service-for Physical-Structure Logical-Service-Components)
"Transfer properties from POS to Service"	(Provide-Service-for Physical-Structure Logical-Service-Components) (Linear-span Physical-Structure #)
"Verify Skeletal Design"	(<Various relations represented in constraint rules>)

Generalizing Topo's models and operators. Building Topo in Heracles was an incremental process in which we shifted attention back and forth between the details of writing metarules and the high-level definition of blackboards and opera tors. Just as we were able to build another program quickly using Neomycin's relations and operators,[8] we conjecture that expert systems similar to Topo can be constructed more quickly than Topo itself. Toward this end, we need to describe Topo's reasoning and representations at a more general level than computer network layout. Topo's blackboards and operators are suitable, for instance, for "social services network configuration." This characterizes a wide variety of potential expert systems that map from a model of a social structure, such as a university or business description, to a

network of services or suppliers, such as a telecommunication network (see Figure 3). We can reuse existing physical-organizational-structure models in expert systems that design different kinds of systems or processes (for example, an insurance policy configuration). In each of these expert systems, we follow similar steps: determine the physical-organizational structure: determine service and local-resource sizing require ments, and finally, place and connect servers and suppliers. The relations and operators shown in Table I provide a useful level of abstraction for building such systems. Put another way, we don't simply give the knowledge engineer the subtasks and metarules used in Topo; we provide the design of the blackboards in terms of the domain relations of Table 1 and a description of what the operators do.

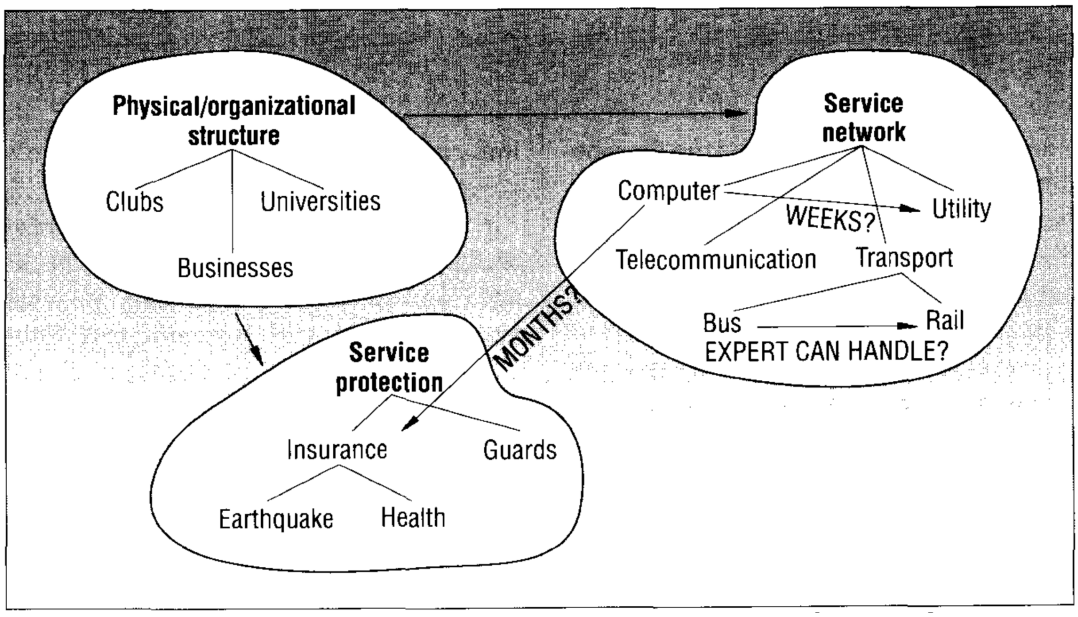

Figure 3. Generalizing TOPO's inference structure. The terms in upper case designate the knowledge-engineering effort required to generalize and specialize existing knowledge bases.

We are also building up a language for describing kinds of systems. Physical and organizational structures differ from service networks, in that the latter involves a distribution of resources rather than a single, roving supplier. For example, Topo's blackboards and operators are probably inadequate for solving the traveling salesman problem. Other expert systems that provide "service protection" also start by constructing models of physical/ organizational structures. For example, an expert system to configure an earthquake insurance policy for a university would develop a model of the university's physical and organizational structure, including a description of buildings and the distribution of work groups. This model will contain new distinctions (relations) that are useful for configuring service protection but not for configuring a service network. This suggests that an ultimate library of system models and operators will be roughly hierarchical, with sharing and specialization of descriptions for different tasks.

Figure 3 shows how we use a classification of system types to describe the relationships among specific expert systems. and what knowledge-engineering effort (upper-case terms) is required to generalize and specialize existing knowledge bases. As always, we use the term "system" very generally; an earthquake insurance policy is a kind of system. We conjecture that it takes much more time to shift between systems of different abstract types (for instance, shifting from service networks to service protection). It should be much easier to reuse blackboards and operators for systems that are of a similar type, such as converting operators developed for bus route configuration to rail configuration.

Figure 3 also illustrates the research task of developing future knowledge acquisition tools. While showing just a small part of the space of potential expert systems, the diagram illustrates the kinds of conversions and adaptations that we expect to find. The system-model-operator metaphor suggests that classifying

ourtools in terms of system models—for example, telecommunication—will facilitate sharing and reuse of representations and inference procedures.

Other task-specific frameworks

Other researchers have advocated the analysis and decomposition of knowledge bases in terms of general, or generic, components. How do these analyses compare? McDermott's role-limiting methods,[10] Chandrasekaran's generic tasks,[2] and the inference operators described here are related as follows:

- A role-limiting problem-solving method is a procedure composed of several inference operators that, through controlled interaction, form a situation-specific model that satisfies task constraints. For example. Neomycin's subtasks together implement a variant of the Mole system's cover-and-differentiate method.
- A subtree in Neomycin's subtask hierarchy (such as explore-and-refine) is packaged and distributed separately in what Chandrasekaran calls a generic task. Subtrees appear in several problem-solving methods.

Thus, from the perspective of the model-construction operator, researchers have pursued different levels of generality in formalizing control knowledge: Single operators in situation-specific models correspond to Heracles-DX subtasks; subprocedures of one or more operators correspond to generic tasks: and a complete inference procedure corresponds to a role-limiting method.

IT IS PRODUCTIVE TO VIEW knowledge engineering as a modeling methodology in which systems are modeled qualitatively in terms of causal, temporal, and spatial relations. From this perspective, control knowledge consists of procedures for constructing situation specific models. Different representations. problem-solving architectures, knowledge acquisition tools, and specific expert systems can then be systematically related in terms of types of relational networks, process models, inference operators, domains, and tasks.

Each time we write a new procedure for interpreting a Heracles representation, we define new relations that classify its constructs. For example, we find that classifications and procedures are defined in terms of each other. When we state control knowledge abstractly, using variables in place of primitive terms, the resulting representation is not domain independent, but domain general; that is, by using spatial, temporal, causal, and subtype distinctions, we can make the language and relations more general than any one system being modeled—a perspective by which all systems can be described.

The system-model-operator paradigm can be used as the basis for analyzing the models and inference procedures in any expert system. It helps us develop new programs, like Topo. but its strength lies especially in helping us relate different research terminology. We can relate blackboards to metarules, Neomycin to Mole, heuristic classification to qualitative process simulation, and generic tasks to role-limiting methods. Put another way, we can now relate representational constructs, expert systems in a given domain, inference methods, knowledge acquisition programs, and reasoning strategies, all from a system-model-operator perspective.

Given any expert system, we can ask, "What are the systems being modeled? What are the structure and process characteristics of this system? What kind of relational network is used to represent these structures and processes? What is the inference procedure for constructing a situation-specific model? How is this model employed by later reasoning phases, evaluated, or conveyed to the user?" Rather than simply asking about a new problem domain, "Is there real-world knowledge that allows classification?" we might ask, "Must the system be modeled as open in its interactions with its environment? Is there a known etiological hierarchy? Are there stages or developmental descriptions in volving trends and frequency of behaviors? What experience have people had with this system in rebuilding, modifying, and assembling it in different situations?" Thus, knowledge engineering is a form of systems analysis that emphasizes the qualitative modeling of processes.

Acknowledgments

Topo was designed and implemented with the help of David Marques at Digital Equipment Corp. Partial support for this research was provided by gifts from DEC and Xerox Corp.

References

1. W. van Melle, *A Domain-Independent System that Aids in Constructing Knowledge-Based Consultation Programs*, doctoral dissertation, Stanford Univ., Palo Alto, Calif., 1980.
2. B. Chandrasekaran, "Expert Systems: Matching Techniques to Tasks," in *AI Applications for Business*, W. Reitman, ed., Ablex Publishing, Norwood, N.J., 1984, pp. 116-132.
3. W.J. Clancey, "Heuristic Classification," *Artificial Intelligence*, Vol. 27, No. 3, Dec. 1985, pp. 289-350.
4. B. Hayes-Roth et al., "Accord: A Framework for a Class of Design Tasks," Tech. Report KSL-88-19, Knowledge Systems Laboratory, Stanford Univ., Palo Alto, Calif., 1988.
5. S. Marcus, *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic Publishers, Norwood, Mass., 1988.
6. M. Musen, "Automated Support for Building and Extending Expert Models," *Machine Learning*, Vol. 4, No.3/4, 1989, pp. 347-377.
7. W.J. Clancey, "Model Construction Operators," to be published in *Artificial Intelligence*.
8. T. Thompson and W.J. Clancey, "A Qualitative Modeling Shell for Process Diagnosis," *IEEE Software*, Vol. 3, No. 2, Mar. 1986, pp. 6-15.
9. McDermott, "R1: A Rule-Based Configurer of Computer Systems," *Artificial Intelligence*, Vol. 19, No. 1, 1982, pp. 39-88.
10. J. McDermott, "Preliminary Steps Toward a Taxonomy of Problem-Solving Methods," in *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, ed., Kluwer Academic Publishers, Norwell, Mass., 1988, pp. 225-256.

William J. Clancey is a senior research scientist at the Institute for Research on Learning. In 1975 he joined the Mycin project, and codeveloped the antibiotic-therapy and question-answering programs. He received his BA in mathematical sciences from Rice University in 1974 and his PhD in computer science from Stanford University in 1979. His current interests focus on computational theories of problem solving that incorporate psychological theories of perception and social theories of knowledge. He is editor-in-chief of AAAI Press, a senior editor of Cognitive Science, coeditor of Interactive Learning Environments, and a member of the editorial board of Artificial Intelligence. He is also a member of AAAI, the Cognitive Science Society, and the American Educational Research Association.

Monique Barbanson is a senior software engineer at Metaphor Computer Systems. Prior to that, she was a member of the research staff at the Institute for Research on Learning. She earned her diplome d'ingenieur de 1'Ecole Centrale in 1984 and her MSc in computer science from Yale University in 1986. Her current research interests include user-centered software design, computer mediated cooperative work, and glass-box systems. She is a member of IEEE Computer Society, ACM, S1GCHI, SIGPlan, and Computer Professionals for Social Responsibility.