SYSTEMIC PRINCIPLES FOR HUMAN-COMPUTER INTERACTION

Mildred L. G. Shaw and Brian R. Gaines

Department of Computer Science, York University,
4700 Keele Street, Downsview, Ontario, Canada M3J 1P3

ABSTRACT

Interest in human-computer interaction (HCI) has existed since the early days of computing, became of high significance in applications as interactive computing came into widespread use in the 1960s, and is fundamental to the fifth generation computing system (FGCS) development program. A variety of styles of HCI have evolved through experience and HCI studies are now in a transition to deeper theoretical models from which design principles can be derived. This paper analyses the systemic principles underlying HCI and develops a model for HCI in terms of a virtual machine hierarchy. Analysis of people in various roles within the hierarchy provides a framework for all aspects of HCI. The technical relations between machines in the hierarchy and the psychological relations between people and machines can both be analysed with the common framework of a category-theoretical model of analogy and emulation.

INTRODUCTION

One of the most obvious areas of application for general systems theory (GST) is computing systems. Information technology has generated some of the most advanced engineering systems of our time. The interaction of people and computers together to undertake tasks beyond the capabilities of either alone presents problems of design, implementation and operation that test all aspects of systems theory. There are the systems philosophy problems of the nature of knowledge and intelligence. There are the systems methodology problems of knowledge acquisition and representation. There are the systems characterization problems of analysing systems with mixed discrete/continuous, asynchronous components. There are systems engineering problems in synthesizing such systems.

Computer science is itself a young discipline with its foundations rooted in pragmatism and achievement rather than methodology and theory. The creation of systems to solve problems has been, and still is, the main driving force behind the advance of the technology. Large-scale systems that impact the operation of society are being developed and put into use with little basis for projecting the consequences (Gaines 1984). Complex systems for human-computer interaction (HCI), involving multiple users, computers, tasks and modalities of interaction, are being implemented for tasks such as command and control, with little basis in experience or theory for understanding their operation.

If system-theoretic methodologies have any power it should be possible to apply them to developing foundations for computing systems that will be meaningful and acceptable to computing professionals, and make immediate practical contributions to computing system design. The challenge is significant not only in terms of computing itself, but also as an exemplar of the problems of modern science and technology. The pace of change is so great that new disciplines are developing without the time to develop secure foundations. The systems approach that distils the structures underlying all disciplines and then instantiates them in the context of the new discipline is vital to control of the modern knowledge environment.

HUMAN-COMPUTER INTERACTION

This paper examines one area of major concern where theoretical foundations are particularly weak, that of HCI, and develops a systemic model on which to base analysis and design. The objectives are:

(a) To develop a systems framework for HCI that makes provision for models of people, computers, and their interaction.
(b) To develop HCI analysis principles that capture all relations between people and computers in human-computer systems, not just those of the user interface, but also those of systems analysis, knowledge engineering, systems design, maintenance, training, and so on.
(c) To develop HCI design principles that facilitate the development of human-computer systems that are locally optimized for individual HCI and globally optimized for overall system performance.

One of the questions always posed about GST when applied to a particular discipline is what contribution does it make that is not already inherent in the discipline? In the context of HCI, one might expect computing science to take models of computers, people and their interaction, and use these to develop an overall model of HCI. The problems and concepts of the subject area seem to reside in the specifics of each constituent, and moving back to take a more abstract and general systems viewpoint seems likely to lose key features of rich and specific situations.

In the context of HCI, the contribution of GST can be seen in terms of such issues as:

(i) Computers and people have a number of common features as communication and control systems. Many specific design rules arise out of these common features.

(ii) Computers and people are both "programmable" in that their behavior may be modified through various channels including "instruction" and "experience." Many specific design rules arise out of these common features.

(iii) The interaction of computers and people may be viewed as that between communicating, goal-seeking systems, with considerations of information flow, competition, co-operation, identification, control, and so on, being highly significant.

(iv) The resulting coupled system has emergent properties not necessarily possessed by either people or computers. HCI involves additional systemic considerations beyond those required for people or computers.

The advantage of treating such issues systemically is the reduction in system complexity that results. A small number of generative principles may be used to account for a high proportion of the system structure and behavior. These principles are modulated in their instantiation by the specific constraints that define "computers" and "people", but the separation of the general from the specific is precisely what gives a simple, correct, useable, and maximally powerful, basis for the analysis and design of specific systems.

Figure 1 illustrates some of these points. It is part of an analysis of the design principles underlying the development of dialog shells, application-independent software packages promoting effective computer-user interaction (Gaines & Shaw 1984b, Shaw & Gaines 1985). This figure was developed from an analysis of the reasons underlying various guidelines in the literature for designing effective HCI. Hansen (1971) at the Fall Joint Computer Conference was the first to develop user engineering principles for interactive systems, and the number of guidelines for dialog engineering has grown over the years. Ours started as a set of 11 rules for programming interactive dialog (Gaines and Facey 1975), growing to 30 recently (Gaines & Shaw 1984a). Williges and Williges (1984) have proposed a set of over 500 guidelines ranging from the general to the specific, and it has become difficult to remember such rules and determine their applicability.

Figure 1 shows that the top level source of guidelines is **systemic**, from general considerations of modeling, control and communication. At the next level the specific features of the **sub-systems**, people and computers, are applied from models derived from the relevant disciplines. At the next level the emergent features of the **interaction** between the sub-systems generate guidelines. At the next level the **style** of the interaction being instantiated leads to guidelines specific to that style. Finally, at the lowest level are guidelines that are specific to a particular application (this level could be extended hierarchically to classes of applications, and so on).

The interaction level is interesting because it is where systemic, person and computer principles come together. For example, the dialog rule **avoid acausality** (Gaines & Facey 1975, Gaines & Shaw 1983) was generated from experience of the ill effects of spontaneous behavior in computing systems on user reactions. This rule has a system-theoretic foundation in that causal modeling systems generate meaningless models of systems with even slight acausalities (Gaines 1976a). However, to apply this principle we have to know that people are causal modelers, and we would not regard the rule as significant unless we knew that time-sharing systems generate apparently random delay distributions.

This paper extends the application of general systems considerations from the limited domain of user-computer interaction to all aspects of the roles and interactions of computing and human systems operating together.

THE VIRTUAL MACHINE HIERARCHY

The concepts in this paper go back to a study of the nature of analog computing

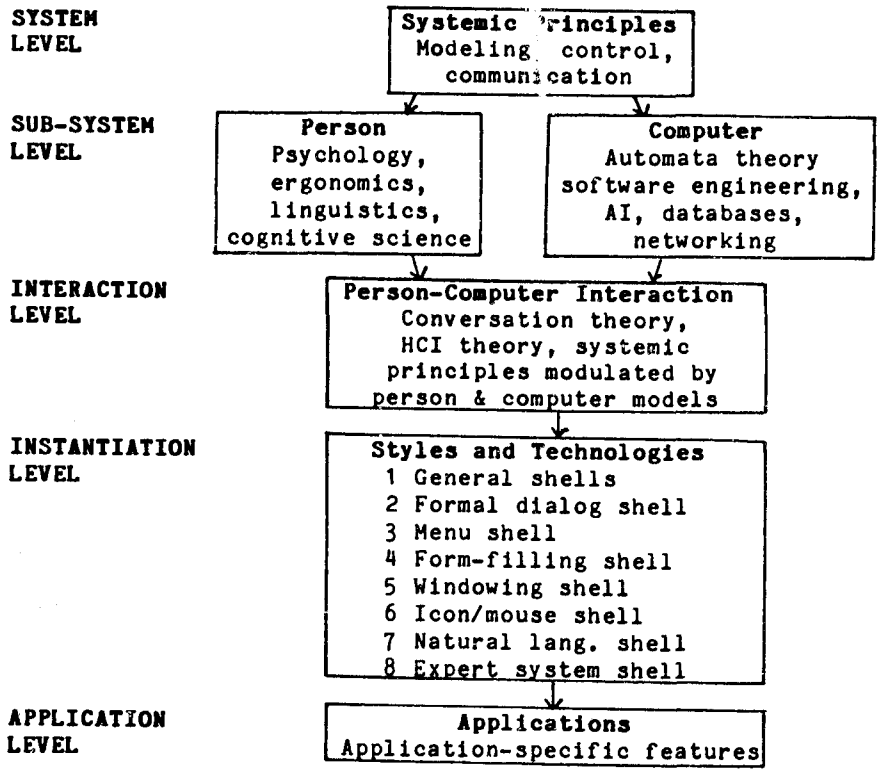

Fig. 1 Levels of dialog rule generation

(Gaines 1968) in which a category-theoretic model of virtual machines was used to formalize what is "natural" in the relation of an analog computer to that which it models. This approach was later generalized to include people and analyse questions about the naturality of programming languages and user interfaces (Gaines 1975). The model has since been used to analyse the relation between hardware engineering and software engineering (Gaines 1977) and the role of the behavioral sciences in programming (Gaines 1979). The category-theoretic model of analogy has also been developed (Gaines & Shaw 1982).

Virtual Machines

The model proposed may be regarded as a systemic analysis of the concept of **virtuality** which has proved an extremely useful one in computer science (Gaines 1975, Weegenaar 1978). We can view a system consisting of a computer together with a usable set of programs either as a programmable machine with certain features together with a program, or as a (programmable) virtual machine (VM) with extended features. This encompasses simple virtuality such as virtual memory where accessing certain locations causes automatic paging, through subroutines regarded not as procedure calls but rather as instruction set extensions, to high-level languages and library systems that appear to their users to have totally different features to the machines on which they run.

Computers may be regarded as algebraic systems applying sequences of functions to discrete spaces and, following Goguen, Thatcher, Wagner and Wright (1973, 1976), we shall treat the abstract form of a computer, or VM, as a category. The category representing digital computers has certain constraints that are not relevant here (the theory of VMs applies to systems other than digital computers). The programming of one VM to run on another is then represented by the existence of a faithful functor from the VM to the one on which it runs (i.e. such that commutative diagrams are preserved).

These functors induce a natural hierarchy on VMs from microcode through to applications (Gaines 1975). In Fig.2, for

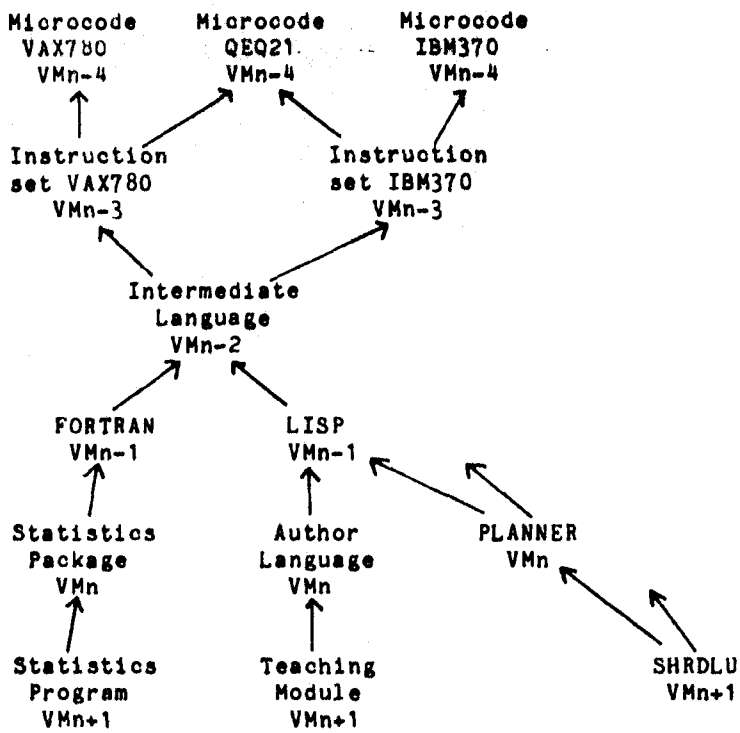

Fig. 2 A virtual machine hierarchy

example, the natural language interface of SHRDLU is a program in PLANNER which is written in LISP which is written in an intermediate language, and so on. The extra arrows indicate that PLANNER could have been written in some language other than LISP and SHRDLU could have been in something other than PLANNER. The remainder of the figure shows relations between different levels of the VM hierarchy, and the role of, for example, an intermediate language in decoupling language translators from actual machines.

Technical Analysis of VM Relations

The technical relations between VM levels, that VMn-1 provides operations suitable for encoding VMn, may be analysed formally in category-theoretic terms using the notion of an **analogy category** (Gaines 1968, 1975, Gaines & Shaw 1982). An analogy is a partial correspondence between two systems and may be represented by a pullback (Pareigis 1970) between the categories representing the systems. Figure 3 shows the semi-lattice of pullbacks representing analogies between two systems (Gaines 1975). Because an analogy is only a partial correspondence it is possible to have trivial analogies, such as null mappings, that do not preserve what is regarded as the essential features of the systems. Hence there is generally a **validation** category that preserves the minimal analogy required to express the "truth" of the relation between the systems (Gaines 1975, Gaines & Shaw 1982). This has faithful functors into every other (richer) analogy. Analogies may be incomparable, e.g. 1 and 3, or one may be richer than another, e.g. 4 is richer than 2. The richer analogy preserves more common features of the relation between the two systems.

Figure 4 shows some analogy relations between virtual machines. VMn has been programmed to run on VMn-1 and emulate some features of a target system. There is an analogy between the target and VMn-1 which is adequate because it is richer than the validation category. The category termed **model** is a valid representation of the target but there is no faithful functor from it into VMn-1. It represents a richer analogy than that actually implemented. This diagram captures the technical relations between VMs at different levels in the hierarchy and the

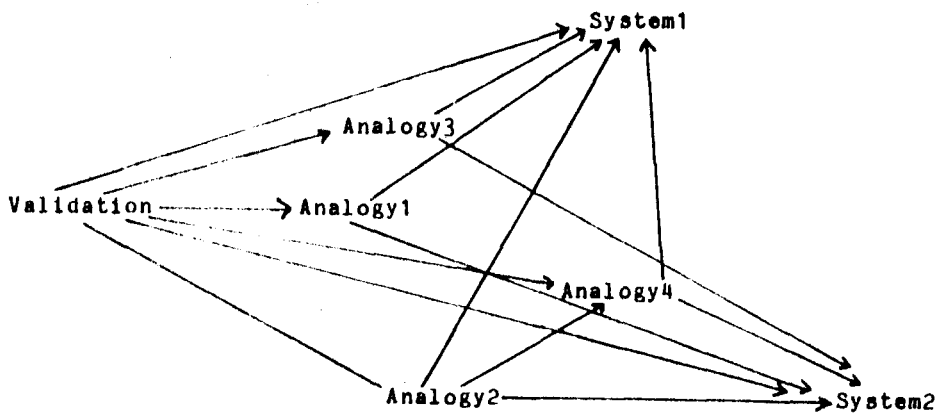

Fig. 3 Analogy categories between two systems

target of programming one VM on another. It shows the role of validation in establishing that the required form of emulation has been achieved. It shows that the VM created may be richer in structure than strictly required, perhaps capturing some problem features that were natural to the implementor but not necessary to its solution.

If the categories are given the structure appropriate to computational systems then further analysis is possible. The naturality of the relations between VM levels may be analysed in terms of computational complexity. If VMn retains all the functionality of VMn-1 but offers access to it in a different way then what has been gained? What is the difference between a high-level language and an assembler? In complexity-theoretic terms we can consider a class of problems together with a measure over that class and examine the expected length and expected run-time of a program for this class/measure on VMn-1 and on VMn. If the expected length and run-time are substantially less for one than the other then the first VM is better orientated to the class of problems. If one is substantially better and the other substantially worse then we have a trade-off between simplicity of program and efficiency of operation.

The Implementor in the VM Hierarchy

These technical analyses miss some major issues, however. The value of an analogy between a VM and its target must be viewed in terms of the cognitive value of that analogy to the programmer. The complexity-theoretic length of a program is not a useful measure if that program is not structured comprehensibly for the implementor, if it is

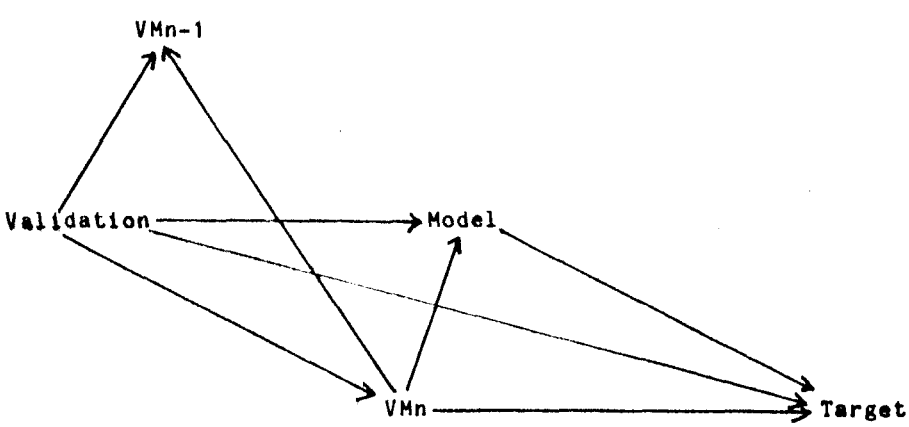

Fig. 4 Analogy categories between target and VMn-1

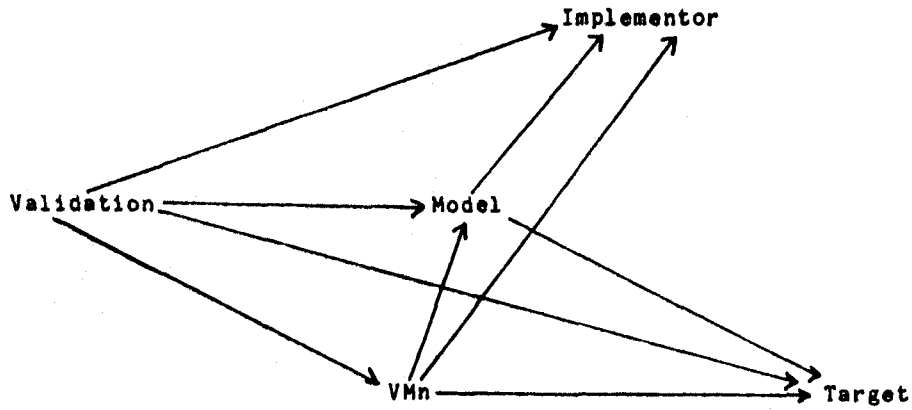

Fig. 5 Analogy categories between target and implementor

not natural for human generation. The decoupling between levels of the VM hierarchy is itself important only in relation to an implementor who need be aware only of the level which he accesses. If, as in Fig. 5, the analogy relations between the cognitive model of the implementor (or implementation team) and the target system are shown, a surprisingly large number of HCI studies may be set in an integrated framework (Gaines 1979).

Figure 5 assumes that the algebraic analysis of the VM hierarchy may be extended to include the programmer if we have adequate category-theoretic models of the relevant cognitive system (Gaines 1975). Some of the formal models being developed in cognitive science show promise of providing this (Weimer & Palermo 1974) such as Scandura's (1973) structural learning. The use of computational models derived from artificial intelligence studies is particularly promising (Bobrow & Collins 1975, Findler 1979, Gentner & Stevens 1983), as are those models based on psychological principles with systemic foundations such as personal construct psychology (Shaw 1980, 1981, Gaines & Shaw 1981, 1984c). The explanation of problems of learning structure from behavior in terms of complexity-theoretic measures (Gaines 1976a, 1976b) suggests that their application to VM-target relations may be valid as a measure of psychological difficulty also.

The cognitive model of the implementor does not necessarily capture all that is in the target system, and may contain additional features not deriving from the target system.

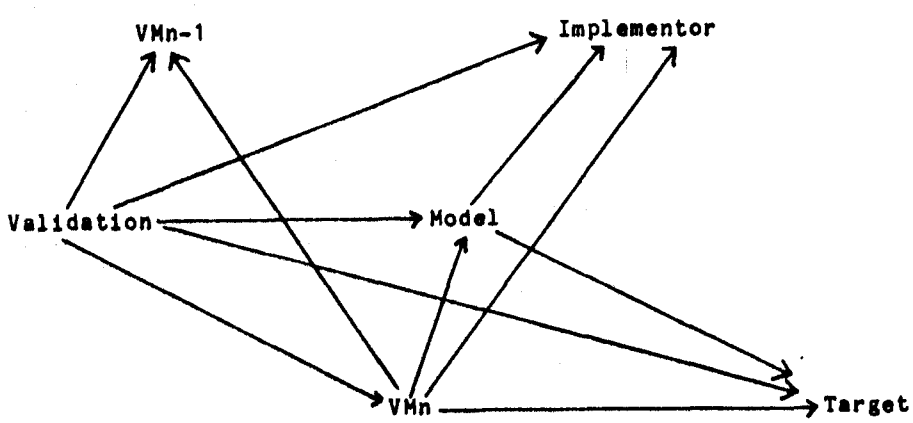

Fig. 6 Analogy categories between target, VMn-1 and implementor

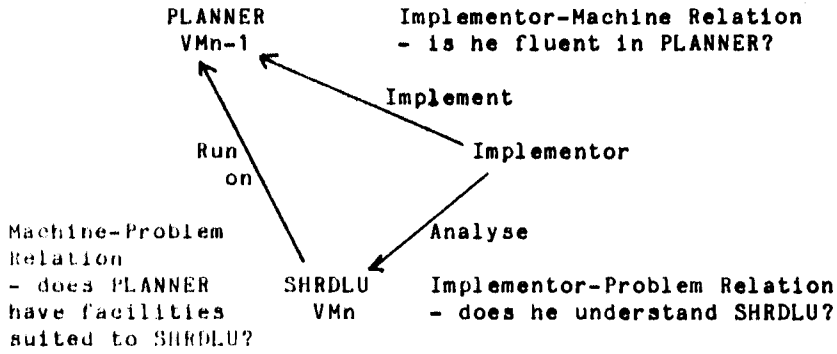

Fig. 7 Three relationships between machine, problem and implementor

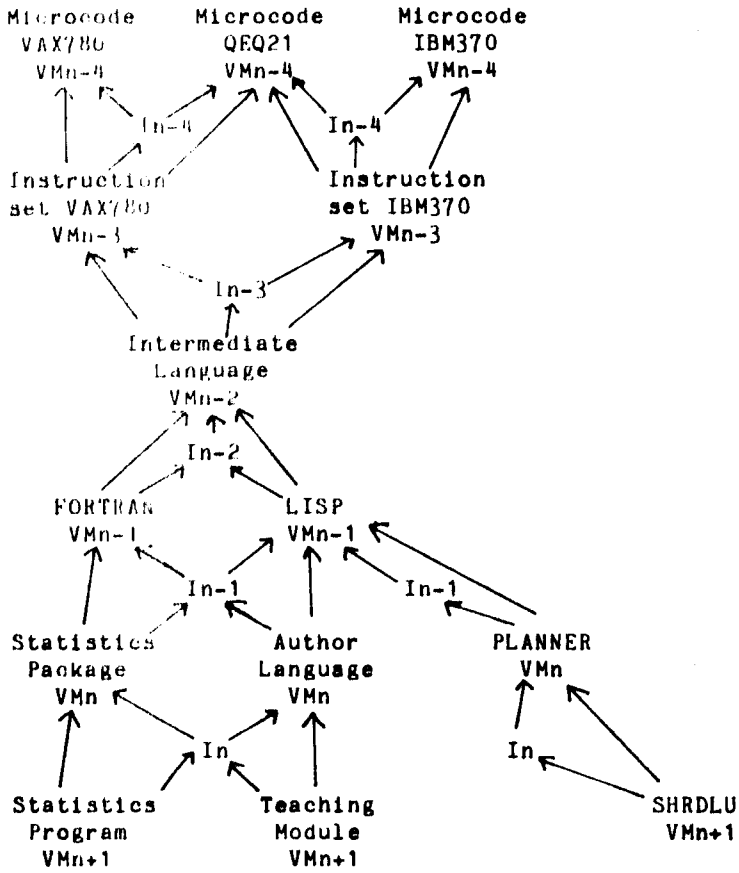

Fig. 8 Virtual machine hierarchy extended with implementors

Thus, it may be represented by an analogy category termed **model** in Fig.5. If the model is adequate there will be a faithful funotor from the validation category to it. The VM implemented may be less rich than the model so that there will be a faithful functor from it to the model category also. Figure 6 shows how the Figs. 4 and 5 may be combined to illustrate the relations between implementor, target and VMs.

Thus, in considering the implementation of a VM there is not just its technical relation to the machine to be programmed at the next level, but also the relation of these two machines to the implementor. Figure 7 shows the kinds of questions that may be posed about these relations and illustrates that an effective implementation involves positive answers to all three questions. The complexity-theoretic relation between VM levels places lower bounds on the length and run-time of programs, but these may be substantially exceeded if the implementor does not understand one or both of the VMs involved. It also enables us to define a problem-orientated language as one in which the problem analysis and the target implementation specification are the same.

Figure 8 shows how the implementors at various levels relate to the VMs at those levels. It makes clear the role of the decoupling between VM levels in large system development in enabling the implementors to be concerned with only part of a very complex system.

Users, Maintainers and Others

The VM hierarchy may also be extended to include the user. Nelson (1980) has emphasized that the user sees the virtual machine not the underlying one, and Smith (1983) has noted this as a design principle of the Xerox Star:

> "if everything in a computer system is visible on a display screen, the display becomes reality."

We tend to treat the user as the end of the VM hierarchy and concentrate on the user interface to the machine. However, a more general viewpoint is to treat the machine plus its user as a new virtual machine, just as we treated the machine plus program as a new VM. We can then see the user+machine VM as an **extension** of either and realize that the key problem is their joint solution of a common task. The HCI within the new VM has a richer significance in relation to the performance of the common task than it does in isolation.

Similar considerations apply to other people with roles in the VM hierarchy such as those responsible for **system** maintenance, documentation, operation, user training, and so on. We can treat them within the framework of cognitive psychology by regarding them as different **P-Individuals** (Pask 1975, Shaw & Gaines 1981) which together make up the team which is the "implementor." A "user" may be regarded as a P-Individual who is built into the VM that is created. He or she is part of the implementation of the target system. A "maintainer" may be regarded as a P-Individual that continues to be concerned with the validation of the VM long-term. A "trainer" may be regarded as the means whereby the implementor "programs" the user.

These considerations allow the full complexity of a large system of VMs and people to be expressed within a single integrated framework. Figure 9 illustrates this type of analysis applied to expert system design. There is a hierarchy of implementors from expert, knowledge engineer, system specifier, system implementor and machine implementor. The roles and sources of information of all those concerned are clearly differentiated:

the **task** is shown as a VM because in general it may not be an end-result but just another VM, or service provided;

the **expert** is someone who can perform the task without any system support;

the **user** is someone who should be able to perform the same task with system support - note that the "user" is actually VMn-1 consisting of the user interacting with VMn-2 - the model allows for the user to perform the task supported by the computer or the computer to perform it supported by the user, or both;

the **knowledge engineer** draws on an **expert model** of how the expert operates, a **task model** of the task, and a **user skills model** of the expertise already possessed by the user, to define a **problem specification**;

the **system specifier** uses the **problem specification** together with a (problem-independent) **human factors** model of the user to define a **system specification** which he can validate;

the **system implementor** draws on the **system specification** and **machine specification** to create VMn-2 by generating code for VMn-3 on which it runs.

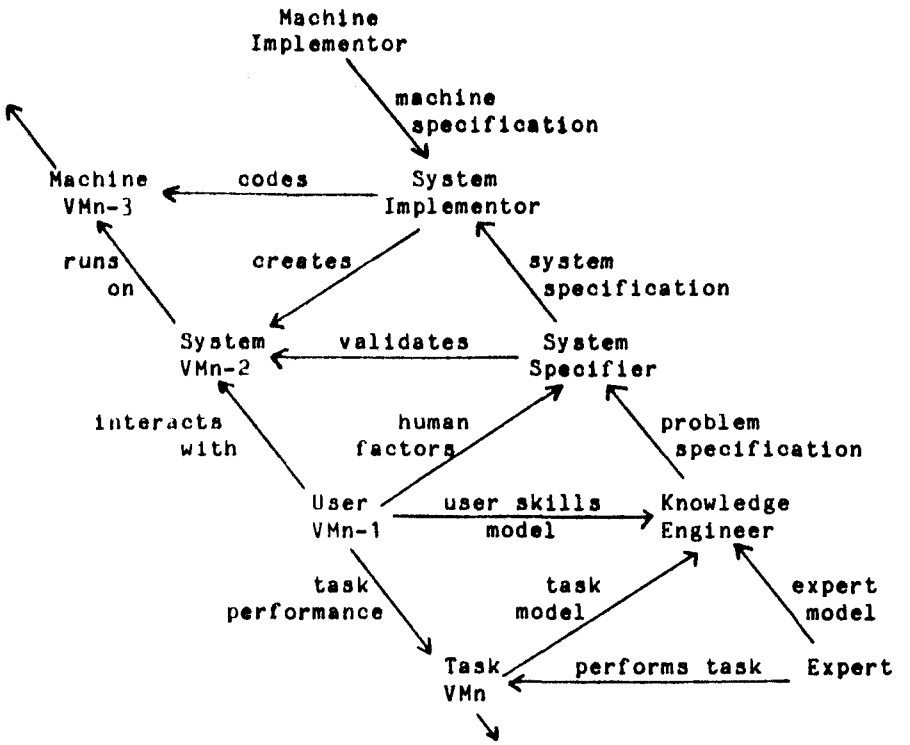

Fig. 9 Expert system design in the virtual machine hierarchy

The significance of documentation can be seen in the flows of information on the right hand side. The separation between task-independent human factors and task-dependent user skills models is also made clear. The role of an expert as adjunct to, or alternative to, direct task analysis may also be seen, and is independent of any particular notion of "expert system"; we have always used expert task performers as sources of system design knowledge. The complexity of human factors considerations in overall system design is apparent from the number of differing roles exhibited by people in this figure. This is further exacerbated when it is noted that the figure is fully recursive in that any person on the right hand side may be provided with computer-based support tools and hence become the user on the left hand side of another such figure.

CONCLUSIONS

The systemic principles underlying HCI have been analysed and a systemic model developed in terms of a virtual machine hierarchy. Analysis of people in various roles within the hierarchy provides a framework for all aspects of HCI. The technical relations between machines in the hierarchy and the psychological relations between people and machines can both be analysed with the common framework of a category-theoretical model of analogy and emulation.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the National Sciences and Engineering Research Council of Canada.

REFERENCES

Bobrow, D.G. & Collins, A.(eds.)
1975 **Representation and Understanding: Studies in Cognitive Science.** New York: Academic Press.

Findler, N.(ed.)
1979 **Associative Networks: Representation of Knowledge by Computers.** New York: Academic Press.

Gaines, B.R.
1968 "Varieties of computer - their applications and inter-relationships." **Proceedings of IFAC Symposium on "Pulse Rate and Pulse Number Signals in Automatic Control"**: 1-16, Budapest.

Gaines, B.R.
1975 "Analogy categories, virtual machines and structured programming." In Goos, G. & Hartmanis, J.(eds.) **GI - 5.Jahrestagung. Lecture Notes in Computer Science 34**: 691-699. Berlin: Springer Verlag.
1976a "On the complexity of causal models." **IEEE Transactions on Systems, Man & Cybernetics SMC-6**: 56-59. No. 1.
1976b "Behaviour/structure transformations under uncertainty." **International Journal of Man-Machine Studies 8**: 337-365. No. 3.
1977 "Hardware engineering and software engineering." **Euromicro 3**: 16-21. No. 2.
1979 "The role of the behavioural sciences in programming." In **Structured Software Development Vol.2**. pp. 57-68. Maidenhead, UK: Infotech International.

Gaines, B.R.
1984 "Perspectives on fifth generation computing." **Oxford Surveys in Information Technology 1**: 1-53.

Gaines, B.R. & Facey, P.V.
1975 "Some experience in interactive system development and application." **Proceedings Institution of Electrical and Electronics Engineers 63**: 894-911.

Gaines, B.R. & Shaw, M.L.G.
1981 "A programme for the development of a systems methodology of knowledge and action." In Reckmeyer, W.J.(ed.) **General Systems Research and Design: Precursors and Futures**. pp. 255-264. Louisville, Kentucky: Society for General Systems Research.
1982 "Analysing analogy." In Trappl, R., Ricciardi, L. & Pask, G.(eds.) **Progress in Cybernetics and Systems Research Vol.IX**. pp. 379-386. Washington: Hemisphere.
1983 "Dialog engineering." In Sime, M. & Coombs, M.J.(eds.) **Designing for Human-Computer Interaction**. pp. 23-53. London: Academic Press.
1984a **The Art of Computer Conversation: A New Medium for Communication**. New Jersey: Prentice Hall.
1984b "Dialog shell design." **Proceedings of INTERACT'84: 1st IFIP Conference on Human-Computer Interaction 1**: 344-349. Amsterdam: Elsevier.
1984c "Foundations of expert systems." In Kacprzyk, J. & Yager, R.R.(eds.) **Management Decision Support Systems Using Fuzzy Sets and Possibility Theory**. Rheinland: ISR.

Gentner, D. & Stevens, A.L.(eds.)
1983 **Mental Models**. Hillsdale, New Jersey: Lawrence Erlbaum.

Goguen, J.A., Thatcher, J.W., Wagner, E.G. & Wright, J.B.
1973 "A junction between computer science and category theory: I, Basic definitions and examples Part 1." **IBM Research Report RC4526**. Yorktown Heights: Thomas J. Watson Res. Lab.
1976 "A junction between computer science and category theory: I, Basic definitions and examples Part 2." **IBM Research Report RC5908**. Yorktown Heights: Thomas J. Watson Res. Lab.

Hansen, W.J.
1971 "User engineering principles for interactive systems." **Proceedings of the Fall Joint Computer Conference 39**: 523-532. New Jersey: AFIPS Press.

Nelson, T.
1980 "Interactive systems and the design of virtuality." **Creative Computing 6**: 56-62. No. 11.

Pareigis, B.
1970 **Categories and Functors**. New York: Academic Press.

Pask, G.
1975 **The Cybernetics of Human Learning and Performance**. London: Hutchinson.

Scandura, J.M.
1973 **Structural Learning. I. Theory and Research**. New York: Gordon & Breach.

Shaw, M.L.G.
1980 **On Becoming a Personal Scientist: Interactive Computer Elicitation of Personal Models of the World**. London: Academic Press.

Shaw, M.L.G.(ed.)
1981 **Recent Advances in Personal Construct Technology**. London: Academic Press.

Shaw, M.L.G. & Gaines, B.R.
1981 "The personal scientist in the community of science." In Reckmeyer, W.J.(ed.) **General Systems Research and Design: Precursors and Futures**. pp. 59-68. Society for General Systems Research.
1985 "Developments in dialog engineering." **Proceedings of National Computer Conference 54**. Arlington, Virginia: AFIPS Press.

Smith, D.C.
1983 "Visibility in the Star user interface." **Proceedings of Graphics Interface '83**: 181. Ottawa: National Research Council of Canada.

Weegenaar, H.J.
1978 "Virtuality and other things like that." **Proceedings of IEEE COMCON CH1388-8/78**: 287-293.

Weimer, W.B. & Palermo, D.S.(eds.)
1974 **Cognition and the Symbolic Process**. Hillsdale, New Jersey: Lawrence Erlbaum.

Williges, B.H. & Williges, R.C.
1984 "Dialogue design considerations for interactive computer systems." In Muckler, F.A.(ed.) **Human Factors Review: 1984**. pp. 167-208. Santa Monica, California: Human Factors Society.