Int. J. Man–Machine Studies (1989) **31**, 1–26

The ECO program construction system: ways of increasing its representational power and their effects on the user interface

DAVE ROBERTSON, ALAN BUNDY, MIKE USCHOLD AND ROBERT MUETZELFELDT†

Department of Artificial Intelligence and † Department of Forestry and Natural Resources, University of Edinburgh, Edinburgh, UK

(Received 15 April 1988 and in revised form 18 July 1988)

There is a growing interest in programs which help users with little experience of computing to construct simulation models. Much recent development work on such systems has utilized comparatively simple mathematical methods (such as System Dynamics) to facilitate the development of a friendly user interface. The problem with these simple modelling languages is that they assume that users have preconceived ideas of the simulation models which they want to build. In the ECO project, which involved the construction and testing of programs to help ecologists build simulation models, it became clear that users could not always adapt their ideas to fit into these mathematical frameworks. They required a more expressive input language in which to describe their modelling problems, rather than being forced directly to specify the programs which solved those problems. However, we found that as the input language became more sophisticated the complexity of the user interface became disproportionally larger. We attempt to clarify the reasons for this phenomenon by comparing the various systems which we built to try to solve this problem. This comparison is facilitated by the use of a sorted logic as a *lingua franca* for the various formalisms used in each system. Our analysis centres around a small number of key characteristics which we use to highlight the strengths and weaknesses of various dialogue techniques.

1. Introduction

Many ecologists would like to construct computer simulations of ecological systems, but are discouraged by the complexity of current simulation languages. They must learn how to program in an available language (a considerable effort); or describe their model to a sympathetic modelling expert who will then implement their model on their behalf. Our goal is to relieve ecologists of this handicap by providing a computer system which is easy to use and which helps them transform their view of a problem from ecological terms, ultimately to working computer programs. A key part of our research has been the design and testing of various mechanisms for conducting dialogues between humans and computers in order to obtain specifications for working programs. This paper contains an analysis of these dialogue systems.

Our central thesis is that, although it is relatively easy to provide interface methods for users familiar with the mathematical formalism upon which a program construction system is based, it is much more difficult to scale up these techniques to

† Correspondence: Dave Robertson, Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, UK.

0020-7373/89/010001 + 00$03.00/0

cater for users with only a vague preconception of the type of program which they require. To make this point, we highlight (in section 2) key characteristics ("dimensions of analysis") which apply to all our experimental systems. Using these dimensions, we describe (in section 3) some interface techniques based on manipulating a simple set of program (in section 3) some interface techniques based on manipulating a simple set of program constructs in a System Dynamics formalism (Forrester, 1961). We demonstrate how this formalism is limiting both as a means of expressing the ecological concepts and as a base for providing effective guidance for naive users. Section 4 describes our attempts to develop techniques which can handle more powerful formalisms while still performing well on one or more of our dimensions of analysis. Our main conclusions are summarized in section 5.

Before becoming involved in an analysis of individual systems, it is useful to define a lingua franca which standardizes the important aspects of all our systems. This helps avoid the common problem of allowing implementation details to obscure the fundamental similarities and differences of each program. We introduce this standard language in section 2.1.

2. Definition of formalism and dimensions of analysis

All the systems constructed during the ECO project (Uschold, Harding, Muetzelfeldt & Bundy, 1986; Muetzelfeldt, Uschold, Bundy, Harding & Robertson, 1985) have the general architecture shown diagrammatically in Fig. 1. Users have some ideas about their domain of application (in our case ecology) and want to explore these using a simulation model. To do this, they must communicate with the *interface system* by supplying a sequence of *input events* (e.g. text input, menu selections). These events will be converted by the interface system into appropriate *input expressions* to the system in some formal language. These expressions must be stored somewhere in the system and we shall call this store a *problem description*. Since ecologists cannot be assumed to be familiar with computers, the *user interface* must help users to provide the correct set of input expressions for describing their modelling problem. The simulation package upon which the final simulation must be run requires a (possibly different) set of *output expressions* which are obtained from input expressions by applying *translation* procedures. The store of output expressions is referred to as a *solution description*. Note that in some systems the input expressions which users supply to the system contain sufficient quantitative detail to be used, without any intervening translation, as output to the simulation package. This is a degenerate case of our general plan, which we discuss more fully in section 3.

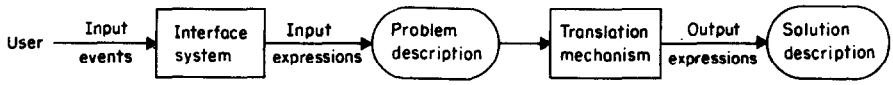

FIG. 1. Schematic plan of Echo

2.1. DESCRIPTION FORMALISM

This paper concentrates on the mechanisms of the user interface. However, the form of the interface is determined to a large extent by the types of input expression

which it must process and this is, in turn, influenced by the types of output expression and the sophistication of the translation mechanism. Therefore, the first step in our analysis is to define a language in which we may represent input and output expressions. We shall then be in a position to compare the mechanisms which manipulate these formal expressions.

We have chosen a sorted logic for this purpose since it covers the full range of expressions used in all our systems without introducing unnecessarily complicated notation. A detailed discussion of this formalism appears in McSkimin & Minker (1979). The expressions in the logic refer to types or objects in a universe of discourse (a set of entities in which the logical constants take values and over which the logical variables range). These entities represent objects and quantities in the real world which users may wish to include in their models. In the ecology domain, for example, there might be hierarchies of types of organism and types of measured quantities. A sample universe of discourse appears in Fig. 2.

The expressions permitted as input or output are represented by functions and predicates defined over certain types. We shall use the following notation:

Words beginning with upper case letters represent variables.
Words beginning with lower case letters represent constants.

$A \stackrel{\text{def}}{=} B$	A is defined to be B.
$\times$	Cross product.
$F \mapsto V$	Mapping from F to V.
A^n	A cross multiplied by itself n times.
$\{E_1, \ldots, E_n\}$	An unordered set of elements E_1 to E_n.
$\langle E_1, \ldots, E_n \rangle$	An ordered set of elements E_1 to E_n.
$\&, \vee, \neg, \rightarrow, \leftrightarrow$	Logical connectives and, or, not, implies, equivalence.
$\forall, \exists$	Universal and existential quantifiers.
$A \in S$	A is a member of set S.
$A = B$	A equals B.
$+, -, *, /$	Arithmetic operators.

For example, the definition:

$$state\_variable \stackrel{\text{def}}{=} object \times time \mapsto quantity$$

states that *state_variable* is defined as a mapping from objects and time to quantities.

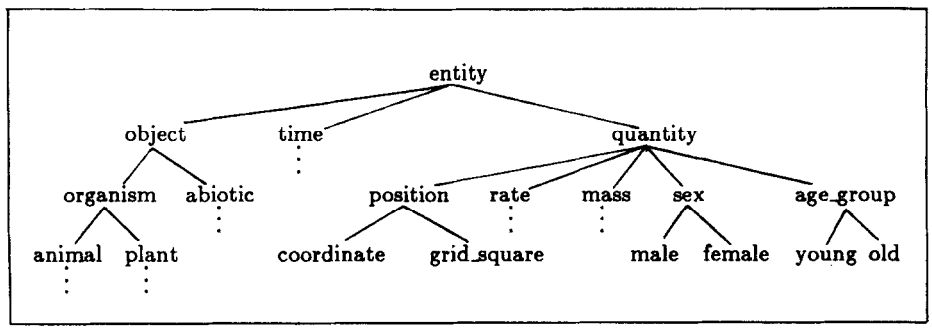

FIG. 2. Universe of discourse.

2.2. DIMENSIONS OF ANALYSIS

A dimension of analysis is an index of the performance of a given system with respect to some key characteristic. In Fig. 3, we introduce a diagrammatic representation in which each dimension appears as a scale representing its complexity (e.g. the guidance dimension ranges from no guidance to the complexities of extensive guidance). We shall use this form of diagram at several points in the subsequent discussion, as a means of summarizing our results. The significance of each dimension is summarized below.

2.2.1. Range of possible input expressions

The capacity of a model construction system to represent a wide variety of relationships in the real world will be determined by the range of different types of expression which users may supply when describing their problems. Since this places an upper limit on the richness of *possible* descriptions, it is desirable that the range of permitted input expressions should be as wide as possible. This dimension therefore varies from a narrow range (with poor expressive capability) to a wide range (with a greater variety of available input expressions).

2.2.2. Range of possible output expressions

In the same way that a wide range of input expressions makes the description of a problem easier, to a wide range of output expressions permits greater flexibility when describing the solution (in our case a simulation model). In reality, the simulation package—upon which the final simulation must be run—is often able to accept only a narrow range of output expressions, perhaps because it has been developed separately or for reasons of efficiency. In such cases the mechanism for translating between input and output must be capable of "compiling" complex input expressions into combinations of simpler output expressions suitable for the simulation package (see section 2.2.6).

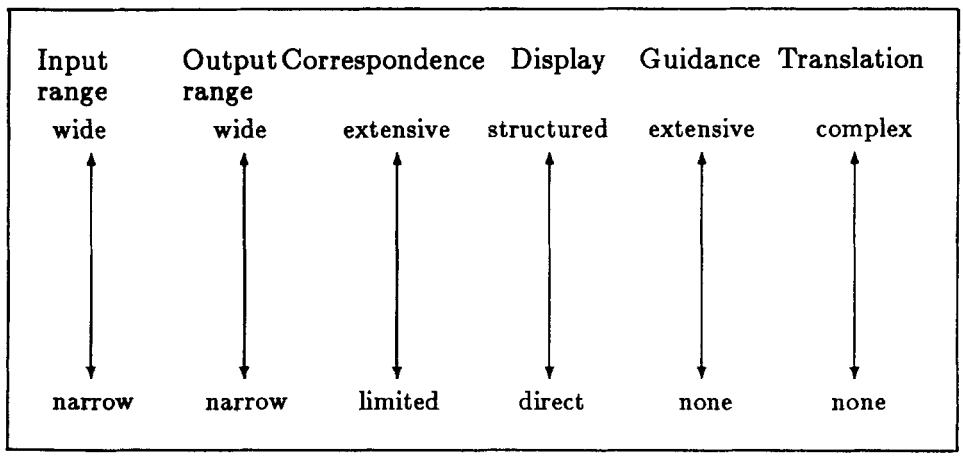

FIG. 3. Dimensions of analysis.

2.2.3. Correspondence between input expressions and users' concepts

Sections 2.2.1 and 2.2.2 are concerned with the *potential* of a system to describe users' models. This potential may not be realized in practice if users are unable to relate the concepts which they have in their minds to the expressions which the system requires as input. Therefore, it is of great advantage for input expressions to correspond closely to the statements which users find "natural". It is difficult to define precisely what constitutes "naturalness", since this may vary between individual users. However, it is possible to weed out systems in which a limited correspondence between users' concepts and input expressions will restrict the number of users to whom they appeal. Thus, our extremes on this dimension are "limited correspondence" (where only a small group of users would find the available input expressions natural to use) and "extensive correspondence" (where the expressions are familiar to a larger user group).

2.2.4. Display of expressions

Users must be able to see the structure of their problem description in order to remind them of what has been said and what remains to be done. They may also require a means of displaying the knowledge base in order to access useful fragments of description. The display provided must be easily interpreted by users. This, in turn, requires that the physical display of expressions should be understandable and (since the number of expressions is potentially large) some means of selectively focusing on parts of the display must be provided. The requirement for an effective display mechanism increases as the range of input expressions is widened, since it becomes impossible to assume that users will remember the meaning of obscure displays if many combinations of symbols are presented. This gives us a dimension of analysis which ranges from direct display, for simple formalisms, to highly structured displays, in cases where the complexity of the formalism makes it necessary to reduce the amount of information which the user sees at a given time.

2.2.5. Guidance methods

Guidance is provided by a system as a means of helping users decide which expressions to include in their problem descriptions. In our experiments, we have concentrated on two forms of guidance. The first of these constrains the *range* of input expressions by recognizing those which will lead to ill-formed models (*completeness checking*) or by detecting expressions which contradict established conventions about "normal" problem descriptions (*consistency checking*). Completeness checks are crucial in systems where input expressions are mapped directly to output (see section 2.2.6) because the translation mechanism will not correct any fatal errors of input. Consistency checks need not be strictly enforced since, although the system may consider them to be justified, a user may have different attitudes. The sophistication of consistency checking is limited by the range of expressions in the knowledge base, since these are what constitute the system's "expertise" in a given domain.

The second form of guidance constrains the *sequence* in which expressions are introduced. Three sources of information may be used by this mechanism. Expressions in the problem description may be used to link the sequence of input to

the developing problem description. Expressions in the knowledge base may be employed as a means of making information about the domain influence the sequence. Special purpose structures may be added to provide more control over the sequence of input. A given system may access information from more than one of these sources.

Our dimension of analysis amalgamates all these criteria into a single index. At one extreme, no guidance may be provided by the system. At the other extreme, extensive guidance of various kinds may be provided.

2.2.6. Degree of translation between input and output expressions

This dimension reflects the amount of work which must be done by the system in order to produce appropriate output expressions from a set of input expressions. This may vary from no translation (where the input expressions are identical to the output expressions required by the simulation package) to a complex translation (if the relationship between input and output expressions is not straightforward).

In section 2.2.2 we noted that some degree of translation between input and output expressions may be necessary if the simulation package which must run users' models can cope with only a subset of the entire range of permissible input expressions. We hinted, in section 2.2.5, that it may also be *desirable* to provide a sophisticated translation mechanism, in order to buffer users from the mathematical complexities of the simulation program.

3. The initial system dynamics based systems

In this section we consider those systems in which users construct models by describing them in a System Dynamics formalism. We begin (in section 3.1) by providing a formal description of the input expressions available to users of these systems. We demonstrate how this formalism limits the way in which users may describe their simulation problem to the system. Given this limitation on one dimension, we consider the effect of various experimental systems on each of the other dimensions of analysis. We shall demonstrate that each of these interface mechanisms relies, for its successful operation, upon the simplicity of the input expressions which it handles. This explains why, when describing interfaces which use a wider range of input expressions (section 4), we have to abandon or adapt these simpler techniques.

3.1. RANGE OF POSSIBLE INPUT EXPRESSIONS

System Dynamics models represent objects in the real world using a collection of "tanks", the contents of which are measured by *state variables*. The value of each state variable may be changed by the input or output of "flows" of material, represented by *processes* attached to state variables. Regulation of the rate of flow of each process is achieved by attaching equations which calculate the rate of that process. Complex networks of equations may be built, introducing *intermediate variables* between linked equations. Variables unconnected to objects in the model may be introduced (*external variables*). All of these model structures may assume different values as time changes during the simulation. Static *constants* are invariant

over time, as are *parameters* of objects. These structures may be represented by the following types of function in the logic, where *object, quantity* and *time* refer to sets of entities in the universe of discourse. We emphasize that these logic definitions are a rational reconstruction of the System Dynamics formalism in a standard language which facilitates comparison with other systems.

$$state\_variable \stackrel{def}{=} object \times time \mapsto quantity$$
$$process \stackrel{def}{=} object^n \times time \mapsto quantity$$
$$parameter \stackrel{def}{=} object \mapsto quantity$$
$$intermediate\_variable \stackrel{def}{=} quantity^n \times object^m \mapsto quantity$$
$$external\_variable \stackrel{def}{=} time \mapsto quantity$$
$$constant \stackrel{def}{=} quantity$$

The computational structure of the model is provided by sets of equation which calculate the appropriate model variables and update the value of each state variable at each time step. Fig. 4 shows an example diagram of a very simple model in standard System Dynamics notation. In the logic, this model is represented as follows:

Objects

$$shp \in animal$$
$$grs \in plant$$

State variables

$$biomass \in organism \times time \mapsto quantity$$

Processes

$$grazing \in animal \times plant \times time \mapsto quantity$$

Constants

$$coefficient \in quantity$$

Formulae

$$decreases(biomass(grs, T), grazing(shp, grs, T))$$
$$increases(biomass(shp, T), grazing(shp, grs, T))$$
$$grazing(shp, grs, T) = coefficient * biomas(shp, T) * biomass(grs, T)$$
$$biomass(grs, 0) = 10$$
$$biomass(shp, 0) = 100$$
$$coefficient = 0{\cdot}2$$

Where: $decreases(X, Y)$ denotes that the value of X at time T will be decremented by the value of Y.
$increases(X, Y)$ denotes that the value of X at time T will be incremented by the value of Y.
T ranges over the integers between initial time 0 to a final time point in the simulation.

We have given an example of how a model can be described using System

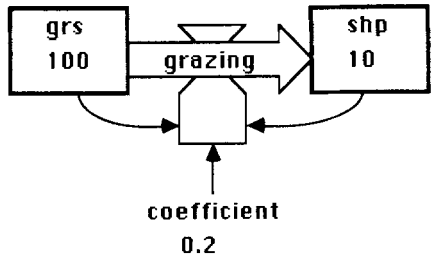

FIG. 4. Simple System Dynamics model.

Dynamics. We now provide some examples of important expressions which it cannot represent, continuing our use of sorted logic to provide rational reconstructions of System Dynamics constructs.

- Quantification of expressions is limited to obligatory universal quantification over time and existential quantification over unassigned function values. All other arguments in expressions must refer to single instances of objects (i.e. they cannot be quantified over a class of objects). This may make some expressions tedious or (worse) impossible for users to supply.

  As an example of tedium, consider a user who wants to say that all the sheep in his/her model have an initial biomass of 10. It would be convenient and more natural to input the expression $\forall S \in sheep\ (biomass(S, 0) = 10)$ but, in pure System Dynamics, this is only possible by adding a separate expression for each object of type *sheep* (i.e. $biomass(shp1, 0) = 10$, $biomass(shp2, 0) = 10$, etc).

  But what if a user wants to say that all objects of type sheep graze some object of type grass at some time $(\forall S \in sheep\ \exists G \in grass\ \exists T \in time\ grazing(S, G, T))$? This could not be expressed in System Dynamics because the existential quantification over grass and time is impermissible.
- Connectives such as conjunction, disjunction and implication are not permitted. Therefore the expression $\forall S \in sheep\ \forall T \in time\ (age(S, T) = 5 \rightarrow biomass(S, T) = 10)$, indicating that all sheep of age 5 have a biomass of 10, could not be used because it contains an implication.
- Predicates are not permitted, only functions which return quantities. For example, a user might want to say that all swallows are migratory: $\forall S \in swallow\ migratory(S)$. There is no facility for handling such statements in System Dynamics.

Why are these particular classes of expression excluded from System Dynamics? The reason is that System Dynamics is a language designed for representing a quantitative *solution* to a modelling problem, based on a particular idealized view of the world (namely that systems can be viewed as collections of tanks connected by flows). This means that users of System Dynamics must mentally convert their view of the real world into the solution formalism in order to describe their model. However, we know from observations that qualitative statements (such as those used in the examples above) are commonly provided by ecologists when describing their modelling problem to human modelling experts. Such statements deserve the title of problem description, since they are intended to describe an ecological

system, rather than specify a program which will simulate it. It is clearly desirable to represent a wider range of input expressions but this extension has important effects on other dimensions of analysis, as we demonstrate in section 4. Meanwhile, we consider how the use of a System Dynamics formalism influences each of our other dimensions of analysis.

3.2. RANGE OF POSSIBLE OUTPUT EXPRESSIONS

Using a System Dynamics formalism, the description of a problem must be made using the *same* expressions which will be used to form the solution to that problem. In section 3.1 we have shown that the range of input expressions is narrow. It follows that the range of output expressions is identically restricted. However, this restriction is less serious for output expressions because these are used simply to run the final program and, as such, can afford to be structured in ways which would seem unnatural and complicated to human eyes. This contrasts with our requirement that input expressions must be capable of being understood by users.

3.3. CORRESPONDENCE BETWEEN INPUT EXPRESSIONS AND USERS' CONCEPTS

Since (from section 3.1) we know that the range of input expressions is narrow and we also know that the range of expressions used by our target user group is quite wide, there is no hope of matching all users' expressions directly to expressions in the problem description. However, we have experimented with two methods of circumventing this problem:

3.3.1. Command language interface

Instead of permitting users directly to insert expressions into the problem description, we can provide a set of commands—resembling ecological statements—which may be expanded, behind the scenes, into a collection of new expressions in the problem description. This principle is used in the Command Language system, in which input strings supplied by the user are used to infer appropriate input expressions. Some annotated examples of these command mappings are supplied below:

If a user types the text string "A,grazes,B", then this causes a grazing process to be created between the biomass attributes of A and B:

$$\forall T \in \textit{time}\ \exists N \in \textit{number}$$
$$\textit{increases}(\textit{biomass}(A, T), \textit{grazing}(A, B, T))$$
$$\& \ \textit{decreases}(\textit{biomass}(B, T), \textit{grazing}(A, B, T))$$
$$\& \ \textit{grazing}(A, B, T) = N$$

If a user types the text string "P,uses,equation1", this defines the value of the process P to be calculated from the quantities associated with equation1 (equation1 relates a process to biomass values):

$$\forall T \in \textit{time}\ \exists A, B \in \textit{object}\ \exists C \in \textit{constant}$$
$$P(A, B, T) = C * \textit{biomass}(A, T) * \textit{biomass}(B, T)$$

If a user types the text string "set,C,N", the value of constant C is N:

$$C = N$$

If a user types the text string "set,A,O,N", the value of attribute A is set to be N at the initial time point (0) in the simulation:

$$A(O, 0) = N$$

We use these mappings to form part of the System Dynamics model shown in Fig. 4 by instantiating them in the following way:

Text input "sheep,grazes,grass" gives:

$$\begin{aligned}&\forall T \in time\ \exists N \in number\\ &increases(biomass(sheep, T), grazing(sheep, grass, T))\\ &\&\ decreases(biomass(grass, T), grazing(sheep, grass, T))\\ &\&\ grazing(sheep, grass, T) = N\end{aligned}$$

Text input "grazing,uses,equation1" gives:

$$\begin{aligned}&\forall T \in time\\ &grazing(sheep, grass, T) = coefficient1 * biomass(sheep, T)\\ &\qquad\qquad * biomass(grass, T)\end{aligned}$$

Text input "set,coefficient1,0·2" gives:

$$coefficient1 = 0{\cdot}2$$

Text input "set,biomass,sheep,10" gives:

$$biomass(sheep, 0) = 10$$

While this method works for simple examples, it does not easily scale up to accommodate complex ecological problems. In particular:

- Domain knowledge is required in order to perform a correct mapping between user input and problem description but this knowledge is supplied in only a primitive form. The first general mapping rule (above) is a good example of this, since *biomass* and *grazing* are names taken from ecology. There is a wide variety of such names—all of which would require an explicit mapping statement.
- The ordered set of input text supplied by the user as a command is very different from the input expression which it implies. This makes it difficult for users to relate their commands to the structural details of the problem description.
- The system requires specific keywords in order to distinguish between commands. These appear as logical constants in the mapping rules given above. Users must be aware of which keywords to use and how to string them together to form valid commands.

3.3.2. Graphical manipulation

If (as we explained in section 3.3.1) it is impractical to distance users from the structure of input expressions, it may be better simply to allow direct manipulation of the problem description—aided by a graphics interface. Such interfaces are found in commercially available modelling systems, of which the STELLA system is one example (Lewis, 1986). Input expressions are each represented by a separate symbol on a bit-map display. These symbols resemble those used in standard System

Dynamics diagrams (Fig. 4) and so are familiar to people who are acquainted with this formalism. Some formal examples of this type of symbol manipulation (taken from one of our prototype systems) are:

If a user picks a "create compartment" symbol and supplies the text string "O,A" then a new attribute is created, representing this attribute A of object O.

$$\forall T \in time\ \exists N \in number\ A(O, T) = N$$

If a user picks a process symbol; two existing compartments (labelled by attribute–object pairs $\langle A1, O1\rangle$ and $\langle A2, O2\rangle$); and supplies the text string "P", a process is created between the two compartments.

$$\begin{aligned}&\forall T \in time\ \exists N \in number\\ &increases(A1(O1, T), P(O1, O2, T))\\ &\&\ decreases(A2(O2, T), P(O1, O2, T))\\ &\&\ P(O1, O2, T) = N\end{aligned}$$

As in the previous example, we can use these mappings to construct part of the example model in Fig. 4:

Event sequence: pick "create compartment" symbol; input text "sheep,biomass" gives:

$$\forall T \in time\ \exists N \in number\ biomass(sheep, T) = N$$

Event sequence: pick "process", pick compartment symbol labelled $\langle biomass, sheep\rangle$, pick compartment symbol $\langle biomass, grass\rangle$; input text "grazing" gives:

$$\begin{aligned}&\forall T \in time\ \exists N \in number\\ &increases(biomass(sheep, T), grazing(sheep, grass, T))\\ &\&\ decreases(biomass(grass, T), grazing(sheep, grass, T))\\ &\&\ grazing(sheep, grass, T) = N\end{aligned}$$

This form of interface works well as a means of constructing System Dynamics models but its success obscures several limitations:

- It is essential that users of this system understand System Dynamics modelling, since there is no provision of domain information to help them. This information could, perhaps, be added to the basic system but this would introduce the domain knowledge problem described in section 3.3.1.
- Since each component of model structure is represented as a separate graphics symbol it is necessary that the number of types of structure should be small, otherwise users will be unable to remember what structure each symbol represents.
- Like the command language system, the user must still think in terms of a *solution* to a modelling problem, described in terms of tanks and flows.

3.4. DISPLAY OF EXPRESSIONS

The graphical manipulation system described in section 3.3.2 provides a simple, built-in display facility, since all objects in the problem description are displayed directly. Users are shown the developing problem description at all times and may

view it as a "real" physical object. There is a close mapping between input expressions and graphics symbol, as can be confirmed by comparing the example problem description in section 3.3.1 with the System Dynamics diagram in Fig. 4.

Even when using this restrictive formalism, a means of selectively focusing on parts of the display is required. This becomes necessary if the problem description becomes large (for complex models) or if users are to be provided with a large number of instances of particular types of expression (e.g. if there is a large data base of equations from which users may select). One flexible method for reducing the display volume is to use a browsing system.

The browsers which we have built use the same general principle, described in Robertson, Muetzelfeldt, Plummer, Uschold & Bundy (1985). They utilize subtype relations in the universe of discourse as a framework for displaying to users the keywords found in expressions stored by the system. Users may move through the nodes in this type hierarchy, seeking out keywords which they consider to be of interest. Since the number of types is likely to be large it is necessary for the interface mechanism to show some small portion of the type hierarchy at any given time. We decided to show a user only the ancestors and subtypes of the node at which he/she is currently positioned.

Having located a type which is of interest, a mechanism must be provided for selecting expressions which refer to that keyword. Consider the following set of expressions, which might be stored in the system's database or (in the sorted logic system of section 4.3.2) form part of the problem description:

Sentence 1: All organisms have some biomass.

$$\forall A \in \textit{organism}\ \forall T \in \textit{time}\ \exists B \in \textit{mass}$$
$$\textit{biomass}(A, T) = B$$

Sentence 2: All plants have some rate of photosynthesis.

$$\forall P \in \textit{plant}\ \forall T \in \textit{time}\ \exists R \in \textit{number}$$
$$\textit{rate-of-photosynthesis}(P, T) = R$$

Sentence 3: All grasses have a rate of photosynthesis of 10.

$$\forall G \in \textit{grass}\ \forall T \in \textit{time}$$
$$\textit{rate-of-photosynthesis}(G, T) = 10$$

Several strategies could be used to isolate some subset of expressions which refer to a given keyword. Three possibilities are:

(1) Select the set of sentences directly containing the keyword.
(e.g. Only sentence 2 refers directly to *plant.*)
(2) Select the set of sentences containing the keyword or its ancestors.
(e.g. Sentences 1 and 2 refer to *plant* and its supertype *organism.*)
(3) Select the set of sentences containing the keyword or its descendants.
(e.g. Sentences 2 and 3 refer to *plant* and its subtype *grass.*)

We have favoured the third option from the list above because it tends to return a smaller set of applicable expressions for keywords near the leaves of the type hierarchy. In tests of this system, users easily accepted the notion of reducing the set

of expressions by selecting more specific types. Note that to help users in this way we have had to incorporate information from the domain of application, in the form of a hierarchy of types.

3.5. GUIDANCE METHODS

If the expressions available to the system are restricted to those of System Dynamics, then the amount of guidance which may be provided for the user is limited. However, the clearly defined structural constraints associated with this formalism make it easy to provide guidance relating to the mathematical completeness of the problem description.

3.5.1. Gap filling

This prototypical system was developed to investigate the possibility of guiding dialogue by referring to the current set of expressions in the problem description. The system comprises three parts:

(1) A mechanism which, for any expression added to the problem description, decides what additional structures need to be added to the agenda of required elements. For example, if the expression $\forall T \in time\ \exists M \in mass\ biomass\ (shp, T) = M$ is introduced then, in System Dynamics, it is essential to know the initial value for biomass of *shp*. Therefore, a value for the variable M in the expression $biomass(shp, 0) = M$ would be sought.
(2) A mechanism which can select a set of options for display to the user from the current set of unspecified structures. The simple algorithm used in this system is to collect all instances of unspecified structures and scan this set for all required structures which perform the same role as the most recently added element. For instance, if the most recently added element was a value for a process and the agenda contained other processes which required values, then those other processes would be displayed to the user. Clearly, this algorithm could be made more sophisticated but serves for demonstration purposes.
(3) A user interface which presents modelling options to the user and prompts for a response. This is text based.

Going beyond completeness checks to include tests for ecological consistency is a difficult problem if only System Dynamics expressions are permissible. For example, if a flow of grazing has been connected to a grass compartment it is likely, though not computationally essential, that a user will want to create a herbivore component to receive the grazing flow. Detecting this sort of ecological requirement is difficult because there are many combinations of System Dynamics structures which might be used to represent any one ecological relationship, depending on the purpose for which the model is to be used. For instance, grazing may be represented as a flow of energy from a plant to a herbivore (in a general grazing model) or as flows of protein and carbohydrate from leaves and stem of a plant to a herbivore's stomach (in a detailed physiological model). Since this system provides no means for users to describe these aspects of their problem, it is unable to provide appropriate ecological suggestions.

3.5.2. Dialogue graph

Guidance concerning the sequence of input events can be provided by defining ordering relations between model manipulation actions. To achieve this, we constructed a dialogue graph, consisting of a set of nodes and arcs. Each node represents some model construction action, while a directed arc from node A to node B indicates that the action at B may be taken after the action at A. Several separate dialogue graphs may be constructed as frameworks for dialogue in different parts of the system. For example, there are dialogue graphs representing model construction actions, model display actions: and a help system. Figure 5 shows a simplified diagram of this system.

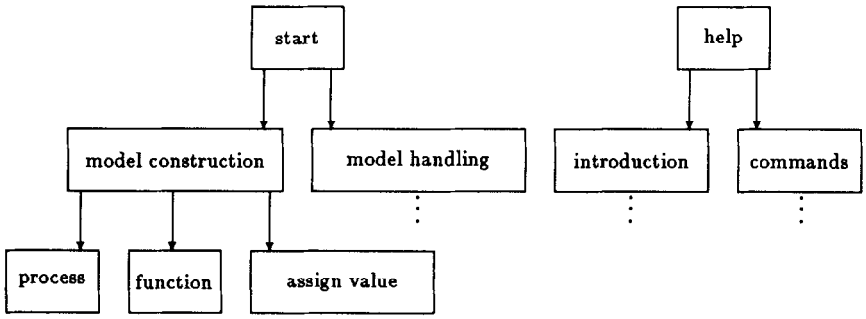

FIG. 5. Dialogue graph system.

When a user visits a node any model construction actions associated with that node will be executed. For example, the procedures necessary to attach a value to a variable are attached to a "set value" node. A small number of keywords are reserved for commands which may be required at any time during the session (e.g. a command to stop the session). Some of these keywords can move a user to a new node in a separate dialogue graph (e.g. the command "help" moves a user to the top node in the help system). This is a convenience measure to avoid cluttering the menu of options displayed to users with items which will always be available to them.

This form of guidance is only practicable if the number of possible actions is small, otherwise the dialogue graph becomes confusing to user and system designer alike. Also, it must be the case that certain actions naturally follow others, otherwise this technique is unhelpful. These requirements can be satisfied using a System Dynamics formalism but would be less likely to apply if the range of input expressions was widened (see section 4).

3.6. DEGREE OF TRANSLATION BETWEEN INPUT AND OUTPUT EXPRESSIONS

Since the System Dynamics notation represents strictly quantitative models, complete with all the detail necessary for them to be run, the expressions supplied as input by the user are identical to those required as output to the simulation package. Therefore, our definition of input expressions also applies to output expressions and there is no need for procedures to translate between input and output expressions.

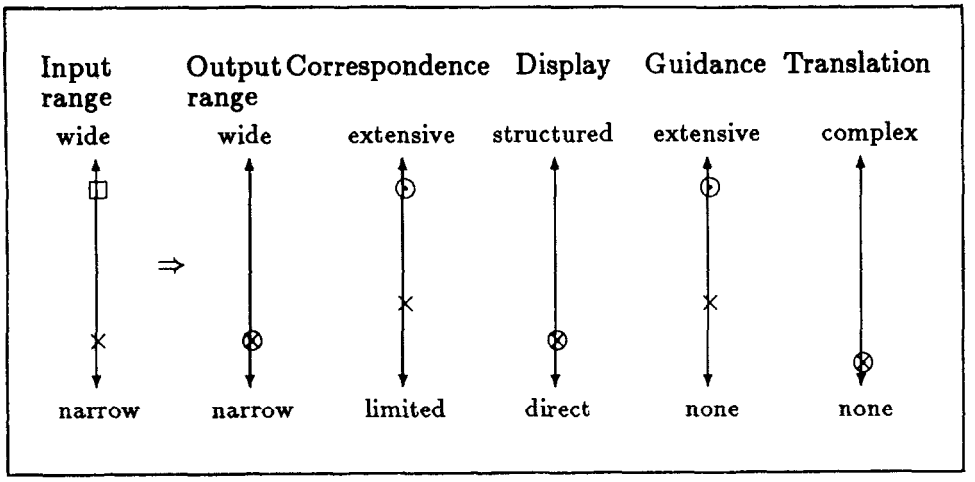

FIG. 6. Performance of system dynamics system on dimensions of analysis.

3.7. ANALYSIS OF DIMENSIONS USING A NARROW RANGE OF INPUT EXPRESSIONS

The experimental systems in this section all occupy the same point on one critical dimension—the range of input expressions which they can handle is limited. This clearly influences the complexity obtained on each of the other dimensions but, less obviously, influences the complexity which is *necessary* to provide a usable interface. We illustrate this point in Fig. 6, which closely resembles Fig. 3 with the addition of three new symbols. First, consider the dimension for range of input expressions, which appears on the left side of the diagram. We have obtained only a narrow input range and indicate this by a × symbol placed near the bottom of the diagram. The fact that we want to obtain a wide input range is indicated by placing a □ symbol near the top. Turning now to the other dimensions: a ⊙ symbol shows the level of complexity necessary for each to support our attained range of input expressions, while a × symbol shows the performance actually achieved.

The systems were of low complexity on all dimensions. However, on three dimensions (range of output expressions, display of expressions and degree of translation) the degree of complexity required for successful operation, given a low range of input expressions, was correspondingly low. Problems occur on two other dimensions. Correspondence between input expressions and user's concepts should be extensive but were, in actuality, limited by the System Dynamics formalism. Extensive guidance is also necessary but was under-supplied in our experimental systems. In section 4, we describe our attempts to counter these problems.

4. Widening the range of input expressions

In section 3.1 we demonstrated that the System Dynamics formalism, upon which all the systems of secton 3 are based, was inadequate for our purposes. It permits only a description of a *solution* to users' modelling problems. We require a formalism in which users can describe the *problem* which their model must address, without having initially to commit themselves to particular simulation algorithms nor to

decisions about how objects and relationships in the problem description will be represented in the final program. The task of the system is then to use this information about the problem to help users construct a solution in the form of a simulation model. In terms of our dimensions of analysis, we can say that the range of input expressions must be widened in order to support more sophisticated guidance methods.

Unfortunately, the interface mechanisms which we described in section 3 all required a narrow range of expressions in order to operate successfully. We are therefore faced with new problems on each dimension:

Correspondence between input expressions and users' concepts will be more difficult to ensure because a wider range of more abstract concepts will be represented. System Dynamics provides a neat analogy between input expressions and a physical structure composed of tanks and flows. This analogy will not hold over a wide range of input expressions (see section 3.1 for examples). Some other means is needed to form a correspondence between input expressions and the real world.

Display of expressions cannot be performed by simply showing a chart of the model structure, since some of the input expressions do not correspond to any single physical structure.

Guidance methods, which could be largely neglected in System Dynamics systems because of the simplicity of the formalism, will be crucially important in systems where the range of input expressions, and therefore the scope for error, is increased. The need for guidance is doubly increased since, by separating problem and solution descriptions, we introduce the need for a complex translation process between problem and solution. Guidance is essential during this phase of model development.

Degree of translation between input and output expressions may be high. In the System Dynamics based systems, the output expressions required by the simulation package are identical to the input expressions. If the same simulation package is used for a system in which a wider range of input expressions is permitted, then a need for translation between input and output expressions will arise.

We now describe some systems which were constructed as experiments in tackling these problems on each dimension.

4.1. RANGE OF POSSIBLE INPUT EXPRESSIONS

The System Dynamics systems of section 3 made no distinction between problem and solution descriptions. We are now considering the consequences of introducing a distinct description of users problems. Clearly, a wide variety of forms of expression could be included in such a description, some of which might require the use of more complicated formalisms than the sorted logic used in this paper. Nevertheless, we chose to use this logic because of its success as a general

specification language (e.g. Balzer & Goldman, 1979; Burstall & Goguen, 1981) and because we have been able to use it to represent formal statements of ecological theory from Niven (1982) and Smith (1974), as well as application models (e.g. Hilborn & Sinclair, 1984). The practical value of this method is that it permits users to input what, to them, look like familiar ecological statements but which the computer can process using the syntax of the underlying logic. The implications of this approach are discussed more fully in section 4.3.

4.2. RANGE OF POSSIBLE OUTPUT EXPRESSIONS

The required range of output expressions will depend on the complexity of the final solution to users' problems and will also be restricted by the capacity of the simulation package to interpret output expressions. It is advantageous to keep the range of output expressions close to that of input expressions, in order to minimize the amount of translation between potentially complex problems and runnable solutions (see section 4.6).

We have experimented with prototype interpreters capable of handling complex solution descriptions but these are computationally expensive to run on today's computers. For non-trivial applications, it will be necessary to minimize the complexity of the solution description, extracting a narrow range of essential computational details from a wide range of input expressions.

4.3. CORRESPONDENCE BETWEEN INPUT EXPRESSIONS AND USERS' CONCEPTS

In the System Dynamics systems of section 3, there was a simple analogy between input expressions and a restricted class of physical objects (namely, tanks and flows). This analogy has been discarded as being too restrictive and we must now introduce some new method for establishing a rapport between user and problem description. We have experimented with two different approaches:

4.3.1. Submodels

One approach is to replace the System Dynamics analogy with another simple convention. However, we know that the input expressions are logically quite complex so we must somehow hide the complex details of expressions inside a "sugar coating". Many users find it natural to represent their models as collections of separate submodels, each of which requires some input data and produces specific output data. This approach to model construction is most appropriate when each submodel represents some identifiable part of an ecosystem (e.g. a submodel representing a typical herbivore). Since ecosystems can be thought of as hierarchies of interacting subsystems, it is useful to enable users to arrange their submodels hierarchically (e.g. a model may consists of a herbivore population submodel and a grass population submodel, with the herbivore population model being divided into several individual herbivore submodels).

This principle has been used in constructing the Submodels system (Muetzelfeldt, Robertson, Uschold & Bundy, 1987), which permits users to build up an executable program by constructing a tree of submodels. The models at the leaves of this tree (called "base models") contain executable code. These base models contain a predefined "chunk" of program for representing some ecological entity and are

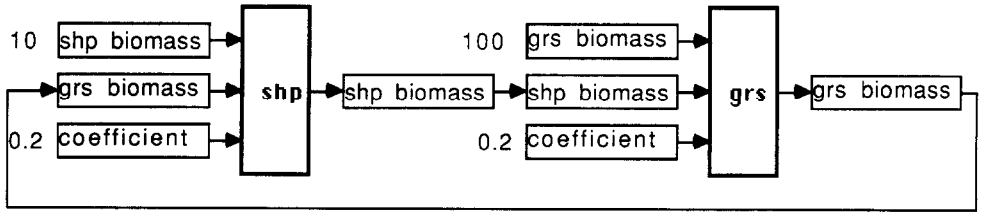

FIG. 7. Simple submodels model.

stored in a database, which may be browsed by the user (see section 3.4). Each new base model introduced into the model is provided with a set of input and output ports. Before running the model all inputs must either be allocated a constant value or obtain a data input from the output of some model. The model tree and data links are created by manipulating symbols in a graphics display (see Fig. 7).

In the logic, all models are simply functions from a specified set of inputs to a given set of outputs. Constructing a tree of models using library functions is achieved by substitution from constituent submodels into the parent built model. Our function definitions are:

$$model \stackrel{\text{def}}{=} input^n \mapsto output^m$$
$$input \stackrel{\text{def}}{=} model\_name \times input\_name \times time \mapsto quantity$$
$$output \stackrel{\text{def}}{=} model\_name \times output\_name \times time \mapsto quantity$$

The model described in section 3.3.1 using the System Dynamics formalism can be defined using Submodels expressions as shown below. A diagram of this model is shown in Fig. 7.

Models

$$animal\_model(input(shp, shp\_biomass, T), input(shp, grs\_biomass, T), input(shp, cpefficient, T)) = output(shp, shp\_biomass, next(T))$$
$$plant\_model(input(grs, grs\_biomass, T), input(grs, shp\_biomass, T), input(grs, coefficient, T)) = output(grs, grs\_biomass, next(T))$$

Equations

$$output(shp, shp\_biomass, next(T)) = input(shp, shp\_biomass, T) + input(shp, grs\_biomass, T) * input(shp, coefficient, T)$$
$$output(grs, grs\_biomass, next(T)) = input(grs, grs\_biomass, T) - input(grs, shp\_biomass, T) * input(grs, coefficient, T)$$
$$input(shp, grs\_biomass, T) = output(grs, grs\_biomass, T)$$
$$input(grs, shp\_biomass, T) = output(shp, shp\_biomass, T)$$
$$input(shp, shp\_biomass, 0) = 10$$
$$input(grs, grs\_biomass, 0) = 100$$
$$input(shp, coefficient, 0) = 0.2$$
$$input(grs, coefficient, 0) = 0.2$$

Where: T ranges over the integers between initial time 0 to a final time point in the simulation.

In this system, correspondence between input expressions and users' concepts is close, provided that users are capable of viewing their systems as hierarchies of interacting systems. It is possible to maintain this analogy for complex models only because the details of complex expressions within base models are hidden from users. Thus, this expressive flexibility is available only to the designers of the base models, not to the users of the system—whose only means of controlling model structure is by creating data flows between models.

4.3.2. Sorted logic sentence editor

If hiding the details of complex input expressions is not a viable option, then a contrasting alternative approach is to ensure that the forms of permitted input expressions closely correspond to commonly used ecological statements and provide a mechanism by which users may manipulate these statements. This approach was the basis for a sorted logic sentence editor, in which a large collection of logic sentences representing possible ecological statements are stored in the system knowledge base. The types referred to in these sentences are at their most general (e.g. "All animals are predators" is the most general predator statement which a user could make in a particular model). Thus, the statements in the knowledge base may be thought of as constituting a pool of template logic sentences from which the users must select those applicable to their models and restrict them to the appropriate types of object. Using a high level specification poses three problems:

- Since the high level statements represent information about an ill defined ecological domain, there is no way of ensuring that all possible ecological statements are capable of being handled by the system. To avoid confusion between what users expect the system to be capable of representing and what it actually can represent, users must be able to examine the knowledge base to see if the system is able to represent their ideas.
- The logic sentences represent ecological statements but the logical notation would be unfamiliar to most ecologists. Therefore, a mechanism for displaying sentences in an acceptable format is required.
- Users must be able to restrict the types of object referred to by selected sentences and, in some cases, to construct new complex sentences from simpler templates. The mechanism for doing this job must be simple to use.

We next describe the methods which we have used to solve these problems.

Users can gain access to sentence templates by moving between items in the type hierarchy using a browsing mechanism similar to that described in section 3.4 and selecting types which they want to consider. They are then given access to all sentences which refer to those types. From a display of candidate sentences, users can select those sentences which apply to their model and restrict their types, using a menu of editing commands. When satisfied that the currently selected templates match their required description of the model, they may insert them into the problem description. Figure 8 contains an example of this method in action. Consider only the larger of the two windows (the smaller one is referred to in section 4.5.2). Here, a user has, using a browsing system, selected a sentence (number 218) from the system's knowledge base. This sentence is represented

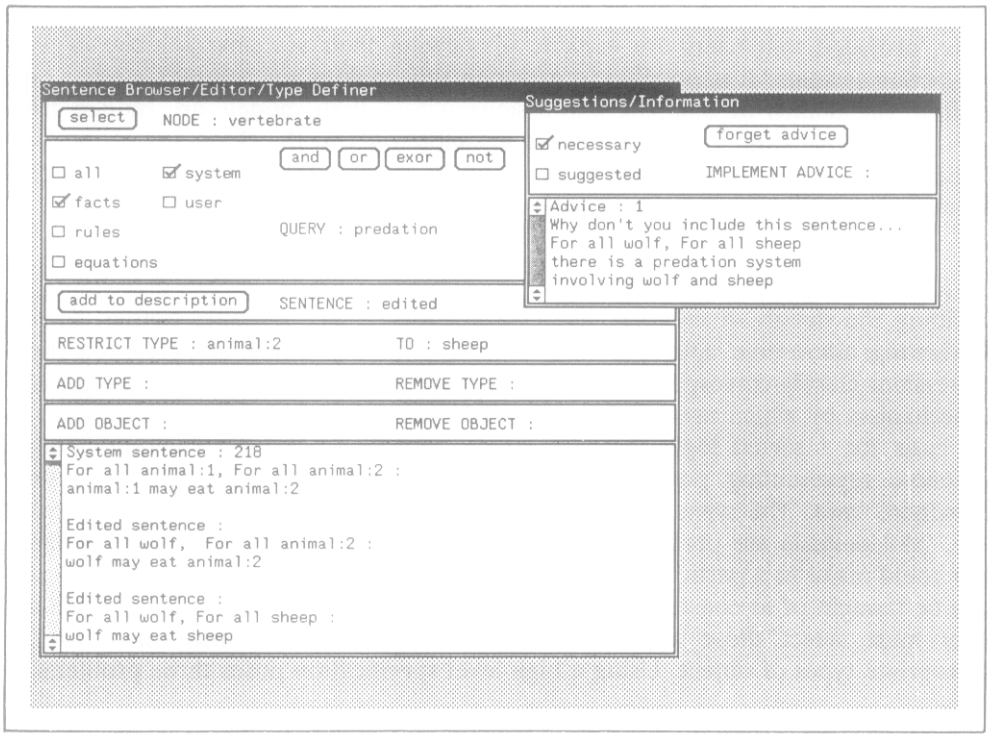

FIG. 8. Sorted logic sentence editor—sample display.

internally as:

$$\forall A \in animal\ \forall B \in animal\ eats(A, B)$$

but has been rendered into stylized English to make the logic more understandable to ecologists. The user has edited this sentence by restricting the type of A to *wolf* and B to *sheep*, forming the expression:

$$\forall A \in wolf\ \forall B \in sheep\ eats(A, B)$$

which has then been added to the problem description.

4.4. DISPLAY OF EXPRESSIONS

The display mechanisms must describe some collection of expressions to users in a form which they can easily recognize and understand. This was easy when the number of different types of expressions was small (see section 3.4) but is now difficult because there is no small set of graphics symbols which can easily capture the richness of all possible input expressions while at the same time being easily interpreted by users. One way of attacking this problem, as we demonstrated in section 4.3.2, is to render each input expression into English text—our justification for this being that we can convey in "natural" language ideas which it would be difficult to represent using graphics symbols. The problem with this approach is that it gives no explicit information about the relationships *between* individual expres-

sions because each expression is represented as a separate English sentence. This may be tolerable when describing user's problems but becomes intolerable when describing solution expressions, since at this stage it is crucial that the expressions mesh together to form a runnable model. This observation suggests that it may be best to represent problem and solution in different ways. The problem must allow a wide range of input expressions, for which the only viable display method may be English text.

4.5. GUIDANCE METHODS

The System Dynamics based systems (section 3) had to provide a high level of completeness checking because the correctness of output expressions depended directly on the correctness of input expressions. These systems, however, did not have the ability to perform sophisticated consistency checks because of the narrow range of expressions which they dealt with. A wider range of input expressions allows sophisticated consistency checking but this extra capability must be harnessed for maximum benefit to users. We compare two mechanisms which we have used to provide this form of help.

4.5.1. Rule based ordering of dialogue

The rule based system was our first attempt to utilize a wide variety of input expressions, representing common ecological statements, as a means of describing the ecological system which is to be simulated. Consider the following "rules of ecological modelling", supplied along with informal text descriptions:

"If there is a grazing system and a predation system
this constitutes a grazing-predation system."
$\forall A, B \in animal\ \forall P \in plant$
$grazing\_system(A, P)\ \&\ predation\_system(B, A) \rightarrow$
$\quad grazing\_predation\_system(B, A, P)$

"If any animal eats any plant then this is a grazing system"
$\forall A \in animal\ \forall P \in plant$
$eats(A, P) \rightarrow grazing\_system(A, P)$

"If any animal eats any other animal then this is a predation system"
$\forall A, B \in animal$
$eats(A, B) \rightarrow predation\_system(A, B)$

These rules form a description framework, connecting general statements, such as *grazing_predation_system*, to definitions of specific interactions, such as *eats*. Further rules could be added to provide more details of the computational structure (linking to equations determining the *eats* relationship, for example). Some of the terms referred to in the rules represent questions which might be asked of the user (if not already established in the knowledge base). These "askable" terms are rendered into a more natural text form before being shown to users. The sequence in which questions are presented is determined by the search strategy used to traverse the rules. Figure 9 shows the sequence in which the system attempts to find instances which satisfy the conditions of the example rule set, using a simple depth-first search strategy, asking low level questions first. A sample of dialogue

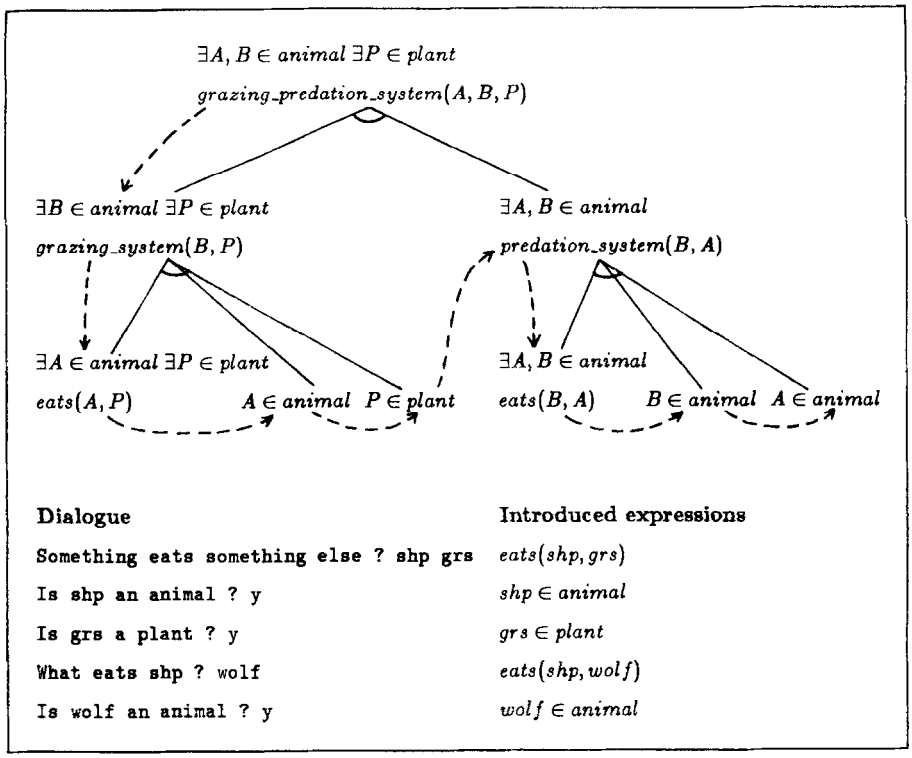

FIG. 9. Simple rule-based dialogue.

produced by the system is included in the diagram and annotated to indicate which parts of the rules were responsible for each question given to the user.

The chief problem with this system is that guidance relies upon a simple search strategy for which the ordering of rules and the structuring of rule antecedents affects the sequence of questions presented to the user. This makes construction of the knowledge base difficult because each rule must be chosen not only to be ecologically consistent but to behave correctly when used procedurally to drive the dialogue.

Also, input of expressions is achieved entirely by accepting or rejecting statements supplied by the system. This may suit naive users but quickly becomes tiresome for those who want to decide for themselves which expressions to input. Skilled users require some method of escaping the incessant prompting supplied by the system.

4.5.2. Guidance by reference to the problem description

Our discussion in section 4.5.1 suggests that it is undesirable to rigidly link the sequence of dialogue to the structure of the system's knowledge base. If this is true, the guidance problem now becomes one of constructing an interpreter which can examine the problem description (containing no dialogue control information) and, in conjunction with any subsidiary information possessed by the system, provide

strategic guidance at a given stage in a modelling session. Our current experiments using this approach are based upon the sorted logic editor described in section 4.3.2, which originally left the onus on each user to decide which input expressions to include in his/her problem description. However, a simple additional mechanism has been added to allow inference of possible new model structures from the set of statements comprising the high level specification. To explain how this mechanism works, consider the following example, which follows from the example of sorted logic sentence editing in section 4.3.2 and completes our description of the display in Fig. 8.

Suppose that a user has added to the problem description the sentence:

$$\forall A \in wolf\ \forall B \in sheep\ eats(A, B)$$

and that the system possesses the rule:

$$\forall A \in animal\ \forall B \in animal$$
$$eats(A, B) \rightarrow predation\_system(A, B)$$

The system's suggestion generator can now apply *modus ponens*, with appropriate substitution of types to suggest the sentence:

$$\forall A \in wolf\ \forall B \in sheep$$
$$predation\_system(A, B)$$

This sentence is rendered into stylized English text and displayed to the user in a "suggestion box" window—the small window in Fig. 8. By referring to the appropriate identification number, the user may get the system to implement this advice. This architecture allows smooth and flexible changes of initiative during the session. It also avoids the perennial problem of ordering the sequence suggestions because the user is allowed to choose which to accept at any time.

We hope in future to elaborate upon the simple inference methods used in our prototype system—for example, by distinguishing general modelling strategies which relate to users' goals for their completed models. For a discussion of these extensions, see Uschold (1986).

4.6. DEGREE OF TRANSLATION BETWEEN INPUT AND OUTPUT EXPRESSIONS

Assuming that some input expressions have been provided by the user, how can these be used to derive a low level computational structure for the model? Previously, we drew attention to the fact that many ecological statements had specific meanings in terms of model structure. For example, if the user provided an input expression corresponding to "All animals have a coordinate location" then there should be some output expressions representing the X and Y coordinates of each organism of type animal. Certain parameters of this structural schema might need to be further specified by the user (e.g. the maximum and minimum values of the X and Y coordinates). Therefore, the problem description may be thought of as constituting information which allows the computer to isolate appropriate low level model structures, which may be quite complex, using formal ecological statements, which are comparatively simple. A more detailed discussion of this process appears in Robertson, Bundy, Uschold & Muetzelfeldt (1987).

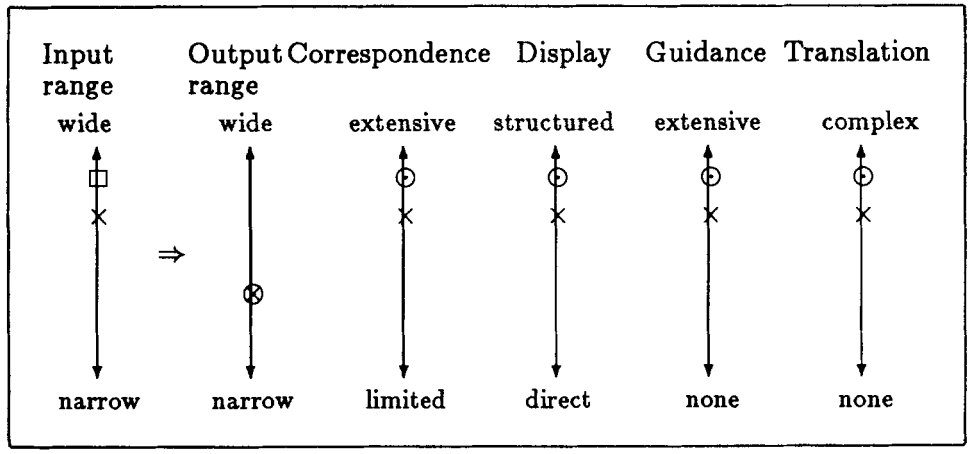

FIG. 10. Dimensions of analysis using a wide range of input expressions.

4.7. ANALYSIS OF DIMENSIONS USING A WIDE RANGE OF INPUT EXPRESSIONS

To summarize our results from this section we provide another diagram of dimensions (shown in Fig. 10) and compare this to a similar diagram constructed for the System Dynamics systems (Fig. 6). We find that the necessary levels of complexity for each dimension (as indicated by the ⊙ symbol) are all high when using a wide range of input expressions. This contrasts markedly with our diagram for the limited range of System Dynamics expressions, where it was sufficient to have low complexity on three dimensions (input of expressions, display of expressions and degree of translation). We have tackled this problem by devising several techniques for enhancing the dimensions, thus moving the × symbols closer to the top of each dimension. However, we are still far short of optimum performance on all dimensions.

5. Conclusion

In previous sections we have given an analysis of the prototype systems which we constructed as part of the ECO project, using logic as a lingua franca in which to describe the important aspects of each system. These analyses were used to construct dimensions of analysis for key aspects of all systems and the response of these dimensions to changes in the range of input expressions has been shown. To conclude, we summarize our results in a single diagram, shown in Fig. 11. The upper row of dimensions are for systems in which the range of input expressions is narrow. Here, the attained complexity on all dimensions is low but adequate for most dimensions. Unfortunately, two dimensions (correspondence between input expressions and users' concepts, and guidance methods) require extension. Extending the range of input expressions provides a basis for improving these dimensions but requires the complexity of other dimensions to be correspondingly increased (as can be be seen from the bottom row of dimensions in the diagram). The moral of this story is that by substantially increasing the range of input expressions in our

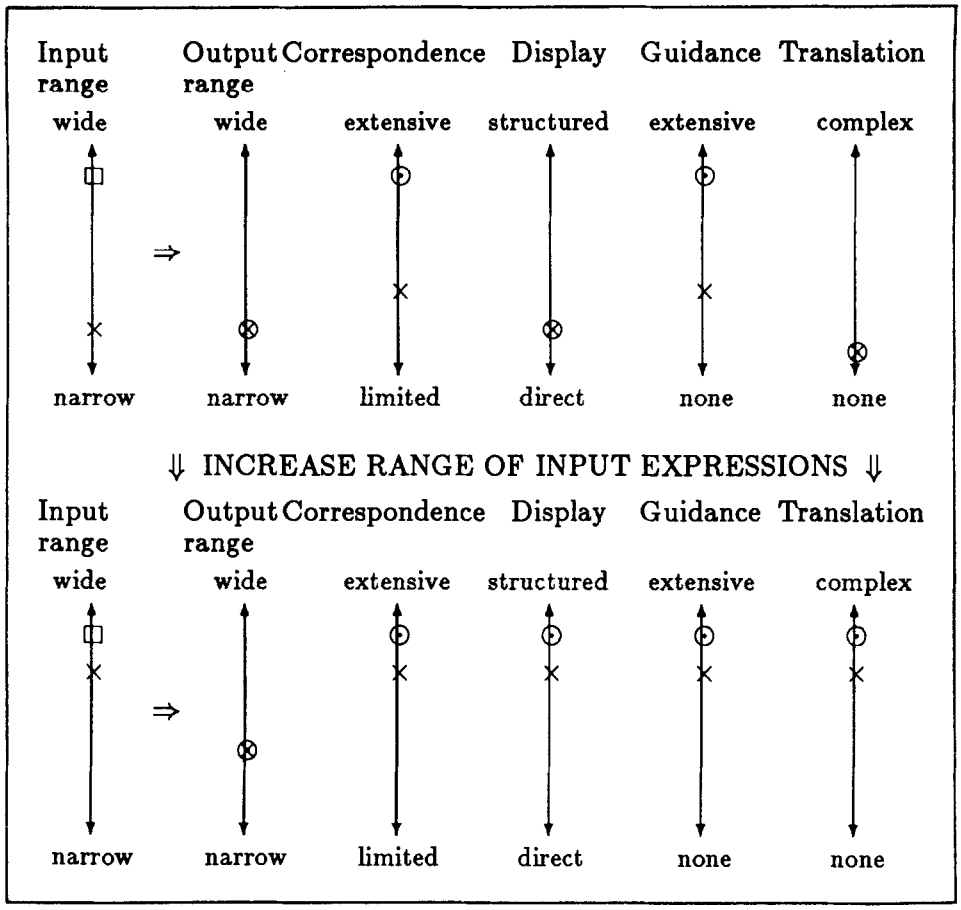

FIG. 11. Effect of increase in range of input expressions.

program construction systems we create a disproportionately large set of new problems in the user interface. Our current efforts to correct this imbalance involve the use of a sorted logic to represent complex input expressions; the provision of an interface to make these expressions easy for users to manipulate; and the provision of sophisticated guidance mechanisms to ensure that a user's ecological problem is adequately represented in the final simulation model.

The research described in this paper has been funded by SERC/Alvey grants GR/C/06226, GR/D/44294 and GR/E/00730. We are grateful to members of the Maths Reasoning Group in the Department of Artificial Intelligence at Edinburgh University for advice and support.

References

NIVEN, B. S. (1982). Formalisation of the basic concepts of animal ecology. *Erktennis*, **17**, 307–320.

ROBERTSON, D., MUETZELFELDT, R., PLUMMER, D., USCHOLD, M. & BUNDY, A. (1985). The ECO browser. In *Expert Systems* **85**, pp. 143–156. Proceedings of the British

Computer Society Specialist Group on Expert Systems. Cambridge: Cambridge University Press.

ROBERTSON, D., BUNDY, A., USCHOLD, M., & MUETZELFELDT, R. (1987). *Synthesis of Simulation Models from High Level Specifications.* Research Paper RP-313, Department of Artificial Intelligence, University of Edinburgh.

SMITH, J. M. (1974). *Models in Ecology.* Cambridge: Cambridge University Press.

USCHOLD, M. (1986). *Computer-Aided Design of Program Specifications in the Domain of Ecological Modelling.* Technical Report DP-35, Department of Artificial Intelligence, University of Edinburgh.

USCHOLD, M., HARDING, N., MUETZELGELDT, R. & BUNDY, A. (1986). An intelligent front end for ecological modelling. In T. O'Shea, Ed. *Advances in Artificial Intelligence.* Amsterdam: Elsevier. Also in Proceedings of ECAI-84, and available from Edinburgh University as Research Paper 223.