COGENT: A Visual Design Environment for Cognitive Modelling

Richard Cooper

Phone: +44 171 631 6211

Fax: +44 171 631 6312

R.Cooper@psyc.bbk.ac.uk

Department of Psychology

Birkbeck College

Malet Street

London WC1E 7HX

John Fox

Phone: +44 171 269 3624

Fax: +44 171 269 3186

jf@acl.lif.icnet.uk

Advanced Computation Laboratory

Imperial Cancer Research Fund

Lincoln's Inn Fields

London WC2A 3PX

June 29th 1997

Please address correspondence to Dr. Richard Cooper

Submitted to *Behavior Research Methods, Instruments, and Computers*

COGENT: A Visual Design Environment for Cognitive Modelling*

Abstract

COGENT is a design environment for modelling cognitive processes and systems. It permits psychologists to construct and test information-processing models of cognition based on traditional box/arrow diagrams. COGENT provides a graphical model editor together with a set of standard types of cognitive module based on familiar theoretical constructs from psychological theory. Models are constructed by selecting appropriate box types, connecting them with appropriate communication links, and configuring the various boxes according to the requirements of the investigator. Once a model has been constructed it may be executed to examine and analyze its behaviour.

1. Introduction: aims and philosophy

Computational modelling is now a well-established approach to theory development within cognitive psychology. However, although many psychologists now have access to sophisticated computational hardware, the practical and technical demands required to make use of systematic modelling techniques can be too onerous for it to be adopted as a routine practice. This paper describes COGENT — Cognitive Objects within a Graphical Environment — a system which aims to allow psychologists to gain the benefits of computational modelling without the heavy investments required by other methods.

COGENT supports computational modelling at two levels. Firstly, it provides an environment in which individual models may be constructed and tested. Equally importantly, however, COGENT provides support for the progressive development of models over time. In an ideal world cognitive modelling would consist of detailed testing of fairly complete theoretical proposals concerning the mechanisms involved in particular tasks or situations. In practice, cognitive theory is highly incomplete and cognitive modelling is generally an iterative process of successive approximations to the desired theory. From this perspective, cognitive models exist within the context of *research programmes*. COGENT provides explicit facilities to support computational research programmes.[1]

Our primary aim in the development of COGENT has been to provide a methodologically rigorous and technically sound tool to simplify the development, understanding, and appreciation of cognitive models. Perhaps the greatest hurdle faced by psychologists approaching cognitive modelling is the apparent gap between the language of theoretical psychology (which typically involves box/arrow diagrams annotated with natural language) and that of cognitive modelling (which is typically C, Lisp or Prolog). This gap would be bridged if theorists could develop models using the familiar box/arrow notation. COGENT addresses this issue by providing a graphical interface which allows the user to "sketch" the processes hypothesised in a model, using standard functional components, and then configure the process details, resulting in a formal but easily understood and executable specification of the required model. For example, one COGENT element is a generalised "buffer"; this has a set of configurable parameters which can be used to determine the properties of a specific buffer in a specific theory, including data capacity, decay rate, retrieval order, etc..

---

*Initial work on COGENT was supported by JCI grant G9212530. Current work is supported by EPSRC grant GR/L03637. We are grateful to all those who have used COGENT and provided us with essential feedback. We are particularly grateful to Sofka Barreau, our first, and most patient, user.

[1]At present COGENT only provides support for purely computational research programmes, but we take seriously the issue of mixed laboratory/computational research programmes and see this as an area to be addressed in future versions of the system.

COGENT provides a set of standard functional components with a range of precisely defined computational properties. These functional components are encapsulated within boxes. This allows the user to select boxes of different classes for different computational functions. The underlying philosophy here draws on the object-oriented paradigm from computer science. The box classes (which are represented in a COGENT box/arrow diagram by different shaped boxes: see Figure 1) correspond to a set of standard configurable components, including *buffers*, *rule-based processes* and *connectionist networks*. The main box classes are discussed in detail below.

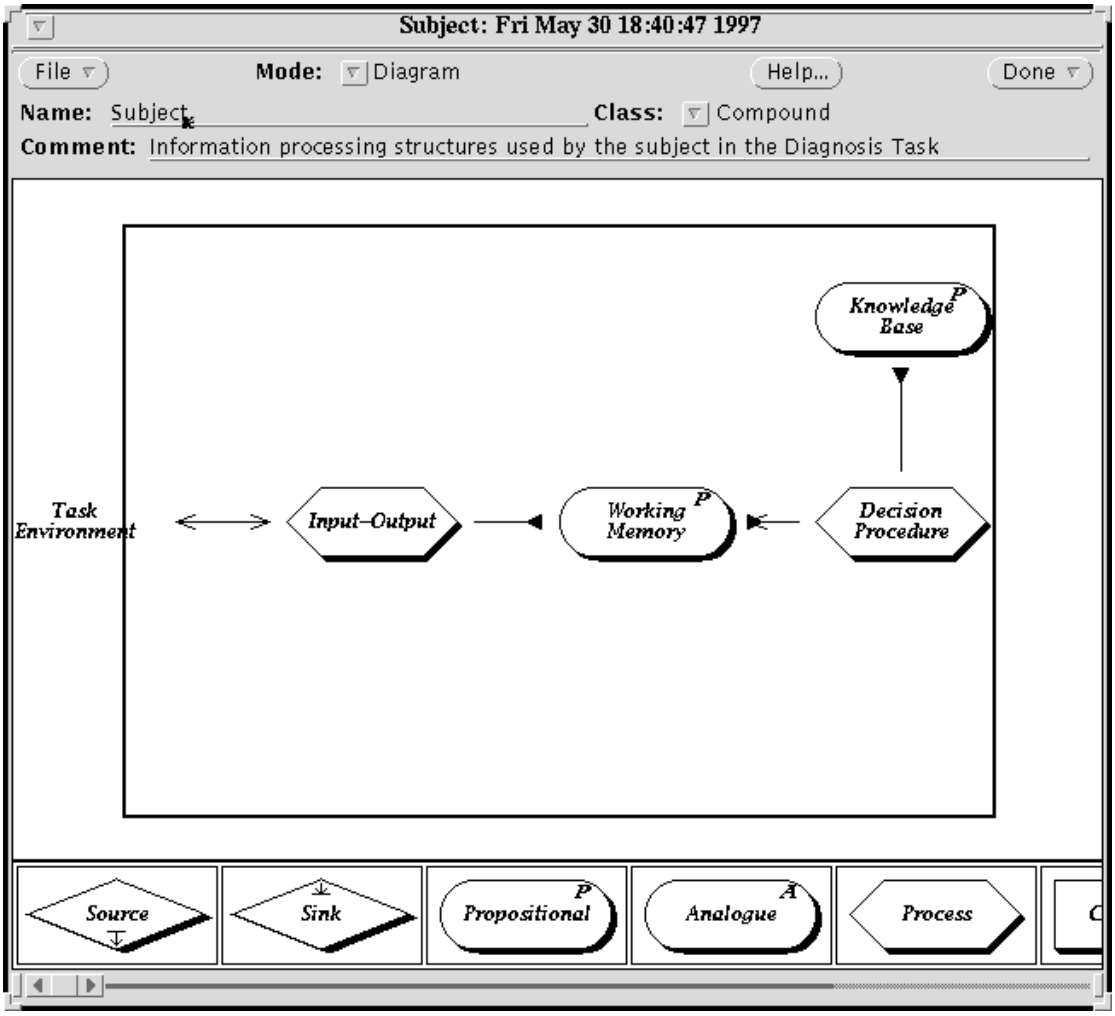

Figure 1: A COGENT box/arrow diagram showing a typical cognitive model of decision making and memory (see text) and part of the palette of standard modules (bottom)

Arrows between boxes simulate communication between the cognitive processes which correspond to those boxes. COGENT supports different communication modes, including *READ* (in which one process reads the data associated with another), *WRITE* (which allows one process to modify the data associated with another), and *SEND* (which allows one process to send information to another).

Beneath the COGENT interface is a well-defined language based on message passing between interacting parallel subprocesses (see Cooper (1995) for details). This language maps directly onto the psychologist's box/arrow notation in which boxes represent functional modules and map to distinct cognitive processes. Once a model is fully specified, the COGENT interpreter can be invoked to run it in order to examine its behaviour and predictions.

The use of standard configurable components has a number of advantages.

Firstly, the existence of standardised definitions of the underlying components reduces potential ambiguities and leads to theoretical statements which are precise and therefore less open to misinterpretation. A common difficulty with much informal theorising in psychology is the lack of clarity and precision. A theory stated in terms of COGENT boxes gains great clarity from the well-understood structure of the components and communication modes.

Secondly, the likelihood of *ad hoc* implementation decisions, or even programming errors, is minimised because the standard components have been used many times by different users and are therefore well validated. This is a serious issue: it can often be difficult to ensure that complex or surprising behaviour exhibited by a computational model is truly of theoretical interest and not the result of some implementation error.

A further issue that often arises in modelling work concerns the relation between psychological theory and practical implementation (cf. Cooper, Fox, Farringdon & Shallice, 1996). Computational models must be "computationally complete" in the sense that they must specify all details necessary for their execution. Psychological theorists often work at a more abstract level. This issue is partially addressed by the provision of box types within COGENT because the psychologist can use the predefined box types to programme at a level which is both computationally complete and psychologically appropriate. By isolating configuration parameters COGENT encourages its users to consider the theoretical justification for decisions both about general (inter)-process organisation and specific parameters and functions.

2. General capabilities

This section provides brief details of some of COGENT's main features; a complete manual is available from the authors on request.

In order to clarify the description of COGENT's components and facilities, we have included a complete example. The model concerns the reasoning processes involved in a medical diagnosis task (Fox, 1980). Subjects attempting this task were required to diagnose a simulated patient's illness, given information about the possible diseases (e.g. laryngitis) and the patient's presenting symptom (e.g., earache). Subjects were able to query the presence/absence of four other possible symptoms before giving their diagnosis, and once a diagnosis was made feedback was given. This allowed subjects to learn the task. The dependent variables were the number of questions asked and the order in which subjects asked those questions. More details of the task can be found in Fox (1980). Details of the COGENT models can be found in Fox & Cooper (in press) and Cooper & Fox (1997).

2.1 Support for research programmes

As noted above, our commitment to the development of models within well-defined research programmes is reflected in COGENT by a built-in "research programme manager". The purpose of this tool is to group related models together, showing the historical relationships between them and allowing them to be manipulated as a single object. When COGENT is started it initially invokes the programme manager. This provides access to all existing research programmes, as well as to a variety of maintenance functions (e.g., for creating and documenting programmes). Individual research programmes can be opened to reveal either a graphical depiction of the programme's history (in terms of an annotated time line) or a text-based programme description. The description must be entered by the user, but the time line is constructed and maintained by COGENT. Each model in the programme is shown as a blob on the time line, with ancestral relationships between models being shown by lines linking the blobs.

Figure 2 shows the history of the Medical Diagnosis research programme at the time of writing. There are six models in this research programme to date. The first is on the left (*Original (1980)*) and the most recent model is the rightmost one (*First learning version*). The diagram shows that this model is based on an earlier model, *Compound version*. It is this earlier model that forms the basis of the remaining examples.

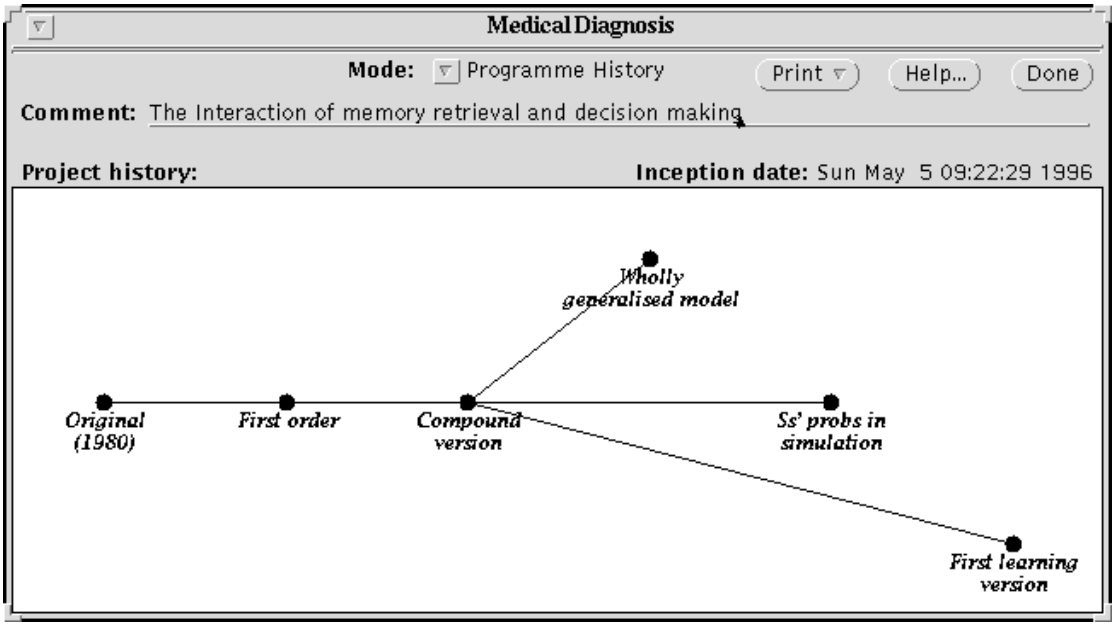

Figure 2: The history of a COGENT project

Selecting a node within the history window causes the model to be opened, revealing the box/arrow diagram representation of the model. Once open, a model may be modified or executed. The history window also allows the creation of new models (normally by copying and modifying an existing model). In this way, the window supports the exploration of variations on possible mechanisms by maintaining the files associated with multiple related models.

2.2 COGENT data representation

COGENT assumes an information processing view of cognition. As such, any COGENT model must deal with the issue of how the information which is to be processed is represented. A common representational language, based on representation within Prolog[2], is employed across all domains. This minimises the computational knowledge required of the user and gives COGENT its domain independence.

Within the representational language, each item of information is represent by a term, which is either atomic (i.e., discrete and unstructured) or complex. Atomic terms include numbers (both integer and real) and strings of lower-case letters. Complex terms take a variety of forms, including lists (represented orthographically as a comma-separated sequence of terms between square brackets) and compounds, which consist of a relation and a bracketed sequence of terms (the relation's *arguments*). A relation is represented as a lower-case string. The following are therefore valid terms of the representation language:

```
                    3.1415
          disease(laryngitis, suspected)
pattern(hepatitis, [told(vomiting, present), told(pyrexia, absent)])
```

The second of these terms is used within the diagnosis model to represent the information that the subject suspects that the patient has laryngitis. The third term is used to represent knowledge gained by the subject during the experiment that if vomiting is present and pyrexia (raised temperature) is absent then hepatitis is probable.

[2]There is no inherent reason why the representation language must be Prolog. It was chosen because a) it is relatively perspicuous; b) it is sufficiently powerful to allow the representation of virtually any form of information; and c) it serves to simplify the underlying implementation.

When specifying a COGENT model the user is required to enter terms in a variety of places (principally in order to configure boxes). COGENT automatically parses all such user input to ensure that all putative terms are valid terms of the representational language. Invalid terms cause COGENT to print a warning message and attempt a correction.

2.3 Box types and their properties

As noted above, COGENT achieves much of its power by providing a set of standard box classes (cognitive modules), instances of which may be used as appropriate within any model. Each instance of a box inherits computational properties from its class (thus, instances of the buffer class inherit the computational properties of buffers). In addition, most classes have a set of properties and an initial state which together allow their computational behaviour to be tailored to any specific use. Once a box has been drawn, it may be opened (by mouse clicking on its icon) to examine and set up or modify its initial state and computational properties. In this section we outline the main classes of box provided by COGENT together with their configurable properties.

2.3.1 Buffers

Many processing models require temporary or permanent storage of information as, for example, in models employing some form of short term or working memory and models requiring some representation of long term knowledge. Buffers may serve either of these purposes: they are boxes in which symbolic or analogue information may be stored.

Buffers are represented in COGENT box/arrow diagrams as *round-ended* oblongs. The model of the subject performing the diagnosis task (see Figure 1) uses two buffers: one for *Working Memory* and one for a *Knowledge Base*. As might be expected, these two buffers are modelled with different storage and retrieval properties.

During execution, buffers contain a set of elements, each of which is represented as a *term* (see examples above). Other boxes may access these elements (provided they are linked to the buffer in question by a *READ* arrow — an arrow with a blunt triangular head), or add or delete elements (provided they are linked to the buffer in question by a *WRITE* arrow — an arrow with a traditional pointed arrow head). The precise behaviour of a buffer (i.e., how it reacts to access and modification) is determined by the properties assigned to it by the designer. These include properties for access order (whether newer or older elements should have priority, or whether all elements should be treated equally), properties for capacity (whether capacity should be limited, and if so what that limit should be and what should happen when the limit is exceeded), and properties for decay (whether elements should spontaneously decay from the buffer, and if so what mathematical function should govern that decay). In principle, further properties could be added, but these properties have proved to be sufficient for all applications to date. The full set of buffer properties, and their values for the *Knowledge Base* of our example, are shown in Figure 3.

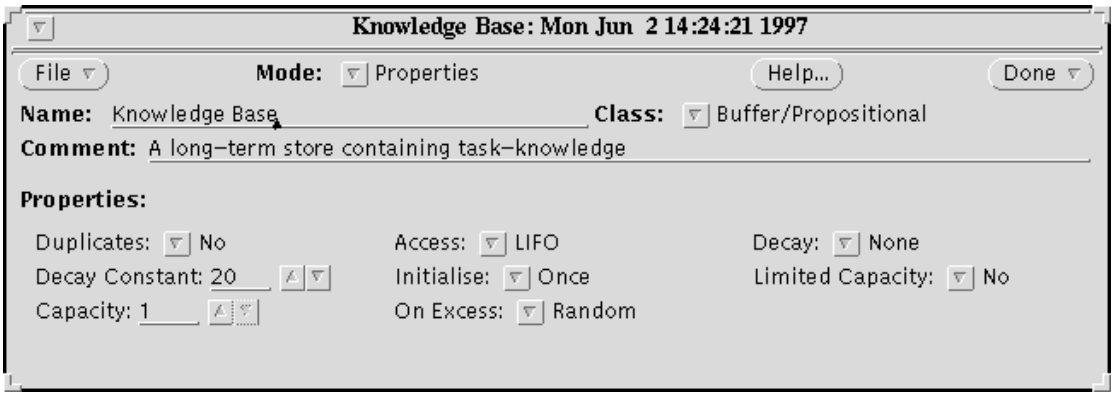

Figure 3: Properties of the *Knowledge Base*

A buffer's behaviour is also determined by its initial contents. In the diagnosis example, *Working Memory* is assumed to be initially empty. If we are modelling a naive subject, then *Knowledge Base* will also be empty. If, however, we are modelling a subject who has already learned the task, *Knowledge Base*, will contain a series of terms representing the subject's acquired beliefs about symptom/disease associations.

2.3.2 Rule-based processes

Rule-based processes are objects that contain a set of rules for processing information. They are represented diagrammatically by hexagons; two such processes are employed in the Medical Diagnosis model (see Figure 1): *Input-Output* and *Decision Procedure*. The behaviour of these boxes is determined primarily by the set of rules contained within them, though the application of rules is tempered by properties specifying firing rate (whether rules should always fire when applicable) and recurrency (whether rules can trigger other rules within the same process). As in the case of buffers (and all COGENT object types), further properties could be added (and will be added if demand is sufficient), but these have proved sufficient for a wide range of cognitive models developed to date.

Rules are specified in a simple condition/action format similar to standard production systems. When a rule's conditions are met it will fire, normally sending (a set of) messages to the buffers or processes which are connected to the process. Individual rules have properties which govern their firing. For example *refracted* rules fire just once for each suitable mapping of terms to variables. *Unrefracted* rules fire every time their conditions are met.[3] Rules may also be triggered (i.e., responsive only to particular messages sent to the process) or autonomous (i.e., continually active and firing whenever their conditions are met).

Figure 4 shows the rules which define the *Input-Output* process in the example. This process is intended to mediate between the processing of the cognitive model and an external experimenter (which, in this model, was also implemented as a set of COGENT boxes).

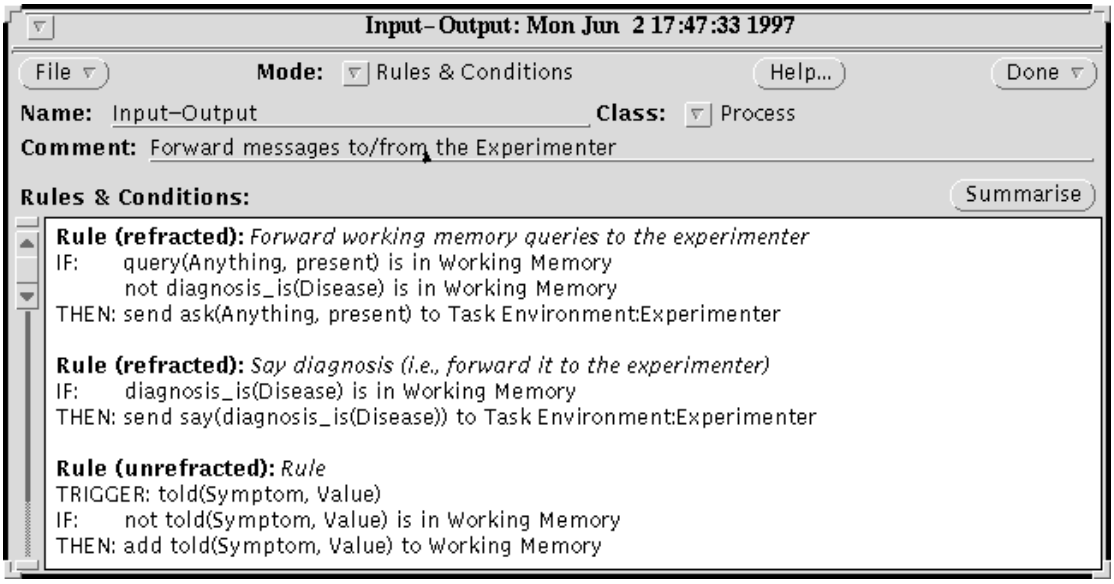

Figure 4: Rules defining the *Input-Output* process

The first rule is refracted and autonomous. It has two conditions: (1) there is a term of the form `query(Anything, present)` in *Working Memory* (where `Anything` is a variable) and (2) there is not a term of the form `diagnosis_is(Disease)` in *Working Memory* (where `Disease`

[3]This notion of refracted rules stems from standard production system modelling, where it is commonplace.

is a variable). If these two conditions are satisfied the rule will fire and send a message of the form `ask(Anything, present)` to the box labelled *Task Environment:Experimenter* (a box not shown on any of the diagrams presented here).

The second rule is similar (though simpler), but the third rule is triggered. This rule only fires when a message of the form `told(Symptom, Value)` is received by *Input-Output*. If and when such a message is received, and if `told(Symptom, Value)` is not already in *Working Memory*, the rule will fire, adding the term `told(Symptom, Value)` to *Working Memory*.

Figure 5 shows the rules which implement the *Decision Procedure*. These rules are more complex, but follow the same basic form as those in the *Input-Output* process. The additional complexities in the rules include the use of qualifiers (`not` and `once`), and the embedding of such qualifiers.

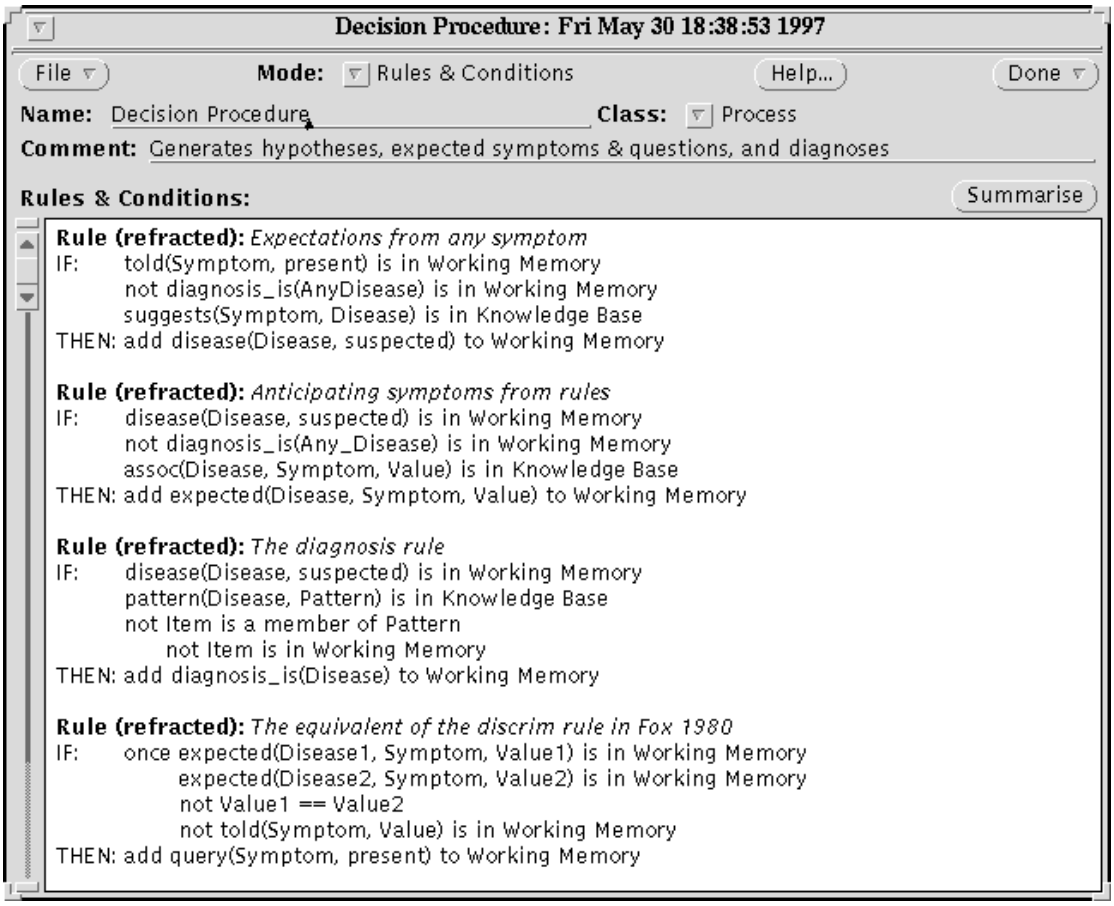

Figure 5: The rules of the *Decision Procedure*

Processes may also contain, in addition to rules, user-defined functions which can be tested or evaluated in a rule's condition. These functions give the underlying language added flexibility at the expense of requiring additional computing knowledge of the user. However, the use of this capability can be seen as an extension of the basic COGENT environment that is only required for advanced models: the Medical Diagnosis model, for example, does not make use of any functions.

2.3.3 Connectionist networks

Network boxes (represented by an icon depicting two rows of nodes) provide a subsymbolic process type within the COGENT environment. At present only two-layered networks are implemented, but capabilities for more complex networks are present. Networks respond to a variety of message types, including training and testing messages (which invoke learning and a response, respectively). Once again, the precise behaviour of the network is determined by a set of standard properties.

Figure 6 shows the full property set associated with any instance of the Network class. The properties specify when the network should be initialised (on each trial/block/subject), how the activations should be initialised (in this case, based on a normal distribution, with mean 0.00 (Weight Parameter A) and standard deviation 0.01 (Weight Parameter B), the learning function and rate (delta-rule learning with a rate of 1.00), the activation function (a sigmoid function centred around 0.00 and with slope 0.25 at its midpoint), the activation ranges (from –1.00 to +1.00), the degree of connectivity (100%) and the width of the input and output vectors. The number and range of parameters here demonstrates the flexibility of the object-oriented approach to box specification.

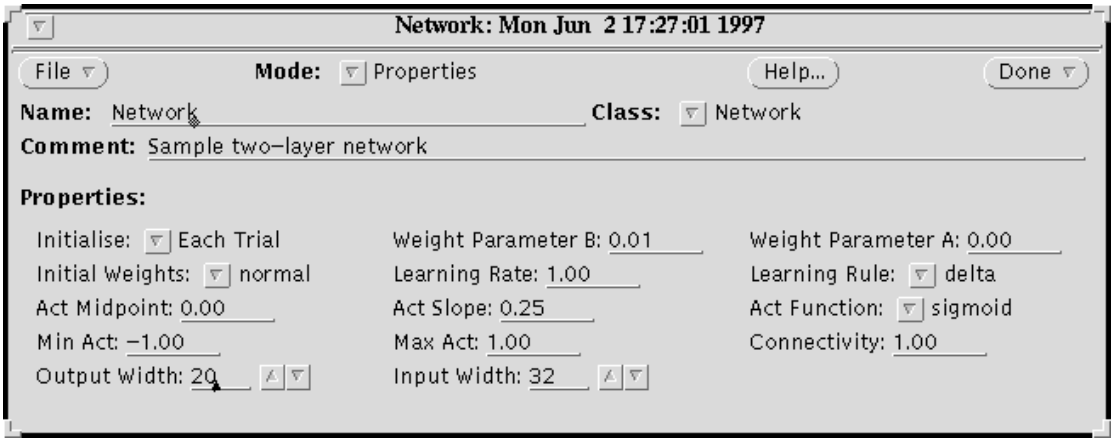

Figure 6: The properties associated with two-layered networks

2.3.4 Compound boxes

Compound boxes (represented as rectangles) are boxes that contain other boxes. In terms of the underlying execution model, they serve no purpose but act merely to bracket sub-components into higher-level functional modules. The contents of a compound box are specified in box/arrow terms (using the same box/arrow editing mechanisms used for complete models), but, as for all COGENT components, these are only visible when the module is opened.

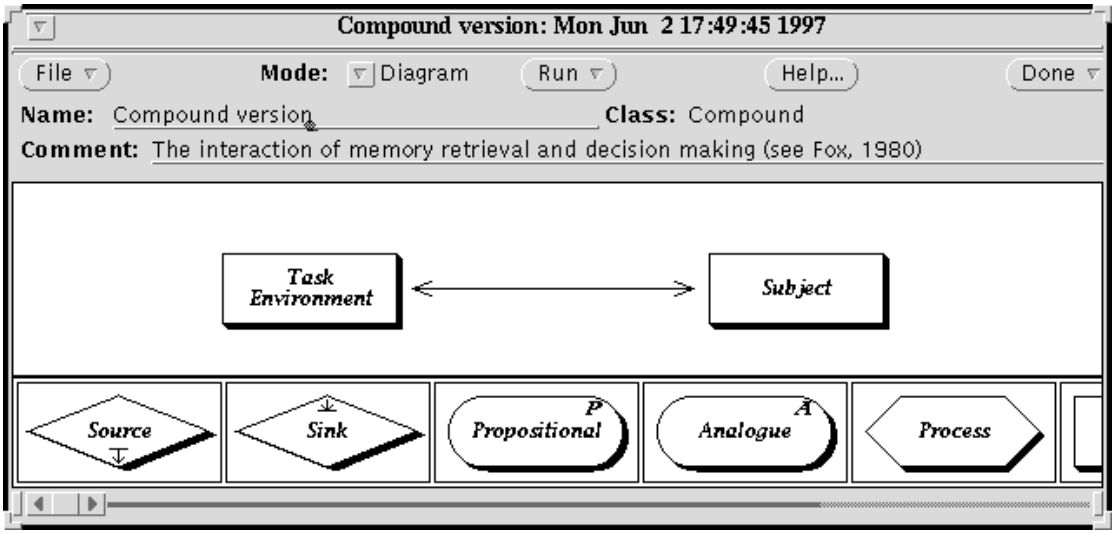

Figure 7: Compound boxes implementing the *Task Environment* and the *Subject*

The example we have been pursuing is actually implemented within a compound box. As noted above, COGENT is used not only to implement the Medical Diagnosis model, but also to implement the experimental environment in which subjects performed the experiment. Figure 7 shows the "top-level" interaction between the *Task Environment* and the *Subject*. At this level, the model consists of

two compound boxes with two-way communication between them. The *Task Environment*, though not described in detail here, uses standard COGENT components to implement stimulus presentation (including randomisation) and data collection.

2.3.5 Data sources

Data sources (which are represented as diamond-shaped boxes) allow input messages to be fed into any box within a model at any time during processing. They are not intended to represent functional elements having any psychological validity, but are provided purely for the purposes of controlling input during simulation runs. Thus, one common use of data sources is to feed the assumed results of perceptual modules to further processes.

Data sources have no special properties to control their behaviour: the behaviour is fully determined by their initial state, which is configured by specifying the sequence of messages which they are required to produce. A special-purpose message editor (described below) is provided to simplify this process.

2.3.6 Data sinks

Data sinks are the output equivalents of data sources. They are employed solely for collecting and storing a model's output. As such their initial state and properties do not affect the behaviour of the model. Data sinks store their contents as a plain text file so that output may be fed into other analysis packages at a later date. Each line of the output file consists of an integer (specifying the cycle number when the output was generated) and a term (corresponding to the output message).

The storage of data within a data sink is controlled by the sink's properties. One property specifies whether output should be stored locally or in a general I/O directory, and a second specifies the filename to be used. A third property specifies whether data should be accumulated over trials, or whether the output file should be reinitialised on each trial. Data sinks, like data sources, are represented by diamond-shaped boxes. A small annotation differentiates the two types.

2.4 Special editors

The behaviour of many COGENT modules is determined by both the modules' properties and their initial states. For each element, the initial state is specified in terms of a set of sub-objects. The type of sub-object depends on the class of the box. Buffers, for example, contain terms, whereas processes contain rules and functions, and compounds contain other modules. Special editors are provided for each type of sub-object. The buffer element editor (shown in Figure 8) is the simplest of these. It allows the specification of an individual buffer element as a term with an accompanying text-based description.

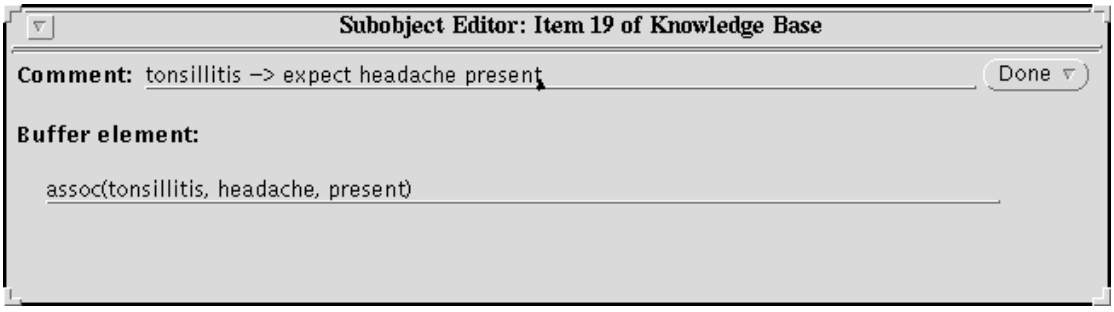

Figure 8: The buffer element editor, showing an element from the *Knowledge Base*

To use the buffer element editor, the user is required to type in a term that should be loaded into the corresponding buffer on initialisation. A one-line free-form text comment describing the term can also be entered. On pressing the Done button, COGENT checks that the term is valid according to its rules of syntax and, if so, adds it to the list of initial elements for the buffer.

The most complex editor is designed for specifying rules within processes. Figure 9 shows the rule editor during the specification of a relatively complex rule. In general terms the editor allows the

specification of a rule in terms of its set of conditions (which may include, for example, matching an element in a buffer that can be read by the process containing the rule) and its set of actions (which may include modifying a buffer or sending a message to some other box). Most rule editing is accomplished through simply selecting appropriate buttons and menu items. Thus, if a rule is required to send a message to a particular box, a menu of possible targets, listing all boxes to which the rule's process has *SEND* arrows, will be automatically constructed (and if there is just one possible target box the message will automatically be directed to that target). It is generally necessary to specify some information directly via terms, but once again such terms are automatically checked for syntax errors before the system accepts them.

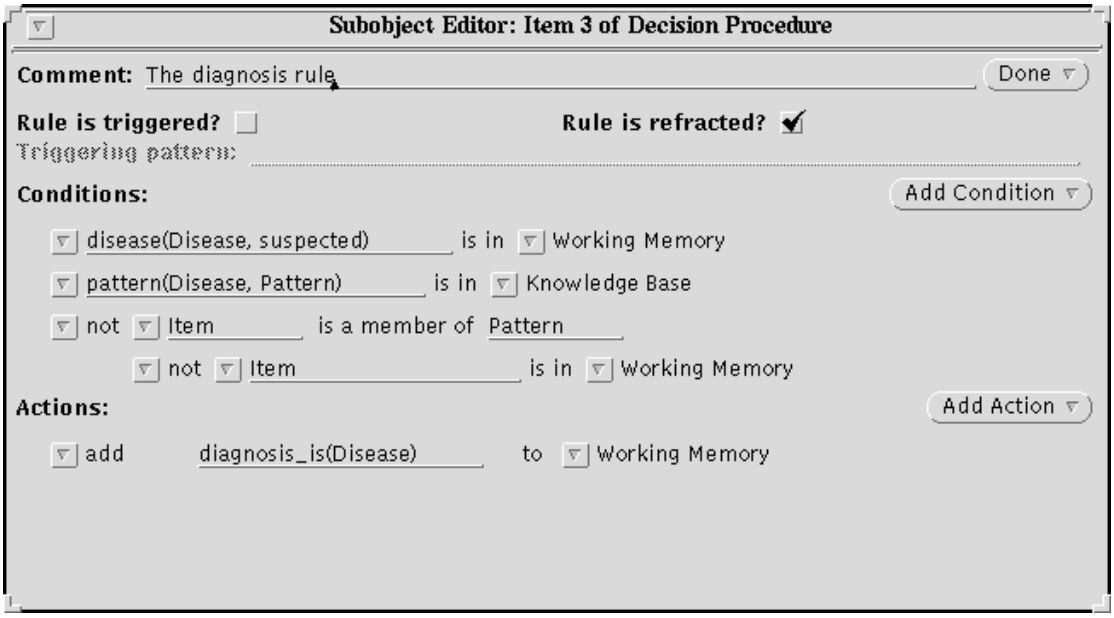

Figure 9: The rule editor, ready to modify the 3rd rule of the *Decision Procedure*

2.5 Model execution

Once a model has been constructed, it may be executed by pressing the Run… button on the model's top-level window. This opens a further window with various execution controls and output facilities. Figure 10 shows this window during the execution of the Medical Diagnosis model.

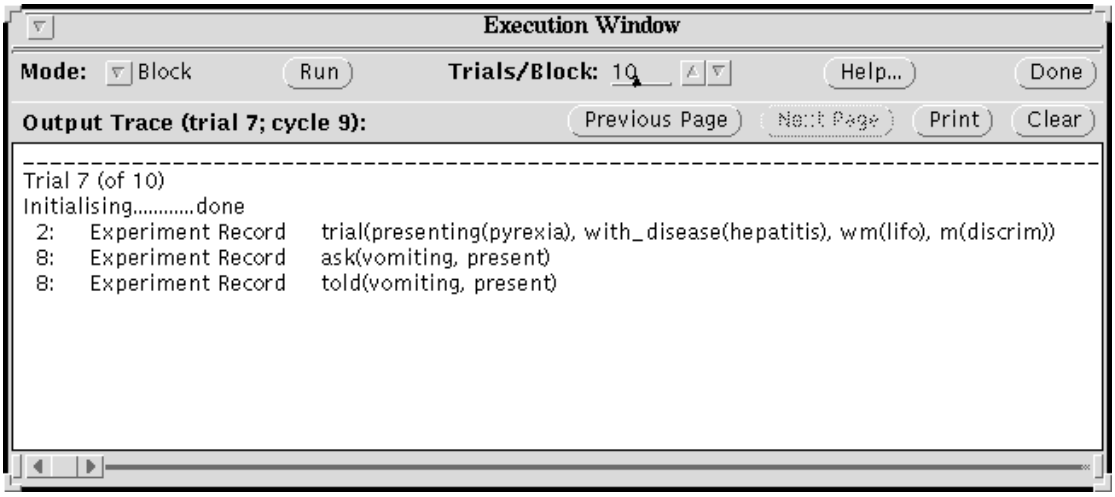

Figure 10: The Run window, showing partial output from one Medical Diagnosis trial

The buttons on the top row of the window allow control over execution (in this case execution of a block of 10 trials is specified). Below these buttons there is a panel giving a window on the model's

output. It shows the current state of processing (in this case cycle 9 of trial 7) and all messages that have been received by data sinks. This lower window can be cleared or printed at any time (including during model execution).

2.5.1 Trials and blocks

A model may be executed in one of two modes. Trial mode is intended for the execution of a single run of a model. This is particularly useful in the development phase during model testing. It is possible to step through individual cycles of the execution model, observing the behaviour and interaction of individual processes, buffers or other modules.

In *block mode*, the user can configure the system to execute a number of trials. Thus, if a model is not entirely deterministic (if, for example, some buffers are randomly accessed or have random decay), Monte Carlo type simulations can be easily performed. This facility is particularly useful in the evaluation phase of modelling. In the case of the Medical Diagnosis model, for example, it has allowed large data-sets to be collected (corresponding to many subjects performing dozens of experimental trials). These data-sets have been analysed and compared with human data to demonstrate that the COGENT model produces a good fit to human performance (Fox & Cooper, in press; Cooper & Fox, 1997).

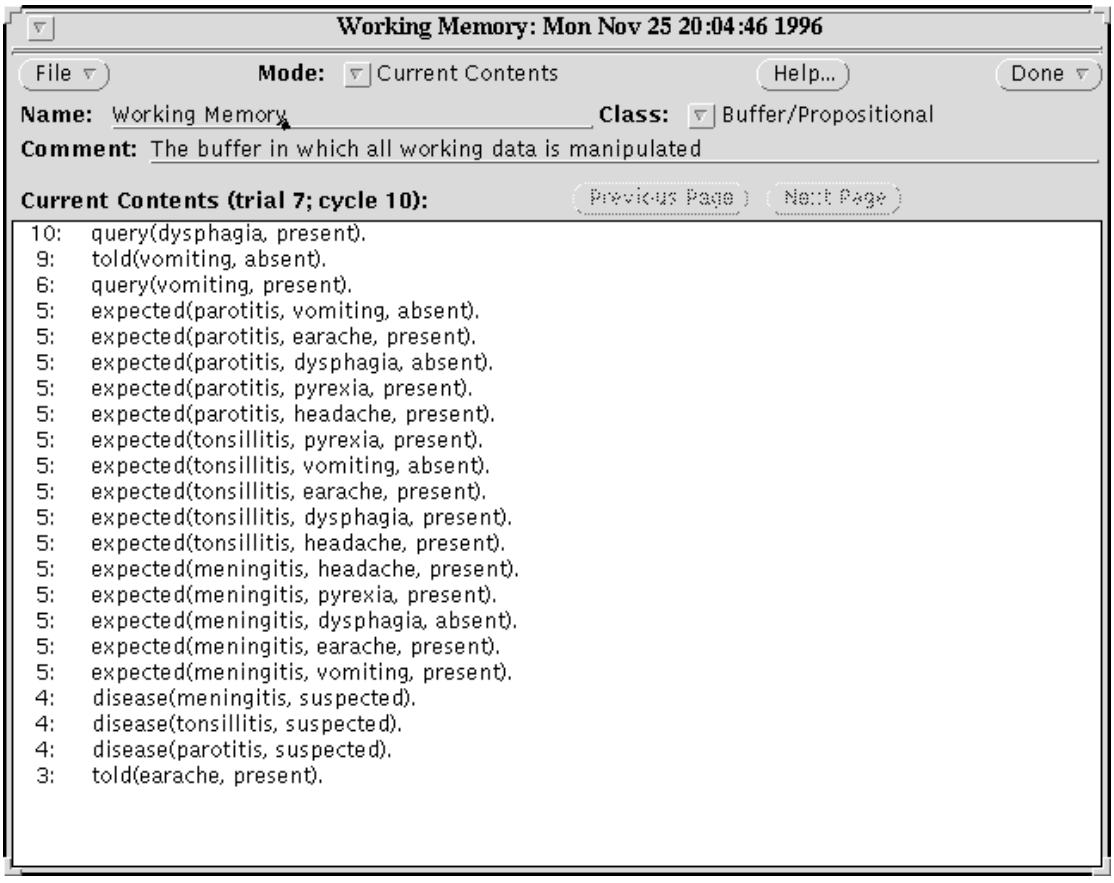

Figure 11: The current state of *Working Memory* in the Diagnosis Model during processing

2.5.2 Ignoring sub-objects

During model development we have found it useful to be able to "hide" parts of a model in order to carry out sensitivity analyses, evaluate functional sub-components of a model, or to experiment with different computational mechanisms. To facilitate this, we have included in COGENT a facility for selectively de-activating model elements. Thus, in the Medical Diagnosis model we have developed separate rules for two strategies which subjects might use (a discrimination strategy and a confirmation strategy). These have both been included in the model, but only one of the rules is activated on any trial. This has allowed the rapid assessment of the behavioural consequences of each

rule without modifying other aspects of the model. Elements from data sources and buffers can also be selectively ignored on different trials.

2.5.3 Current state and message log

When a box's window is opened it shows, by default, the box's initial state. During execution, and particularly during model testing, it is frequently more useful to see the current state (which will change in response to messages received and possibly, in the case of buffers, decay) and the messages being received or generated by the box. A button on each box's window allows switching between these different possibilities.

When a box's window is showing its current state, the window's contents are updated on each processing cycle. In the case of buffers, the current state shows each element actually present in the buffer and the cycle when that element was added to the buffer (see Figure 11). In *trial mode* it is possible to step through individual cycles and watch elements being added and deleted.

In the case of data sources, the current state is a list of messages, the first element of which is processed (and hence removed) on each successive cycle. In the case of data sinks, the current state is also a list of messages, but this list grows during processing as new messages are received by the sink. The current state of a network is its weight matrix. If the network is learning, then this matrix will change as cycling progresses. Other types of boxes (compounds and processes) do not change state throughout processing, and it is not possible to view their current state.

When a COGENT box is set to display its message log, the window shows all messages received and generated by it, together with the cycle number when the message was generated (see Figure 12). The message log is reinitialised on each trial, with messages added to the end of the log in chronological order. If the current state or message log is long, buttons allow scrolling back and forth through the files. These files can be viewed independently of model execution, so it is possible to page back and forth through the files as they are being generated, with updates being displayed automatically.

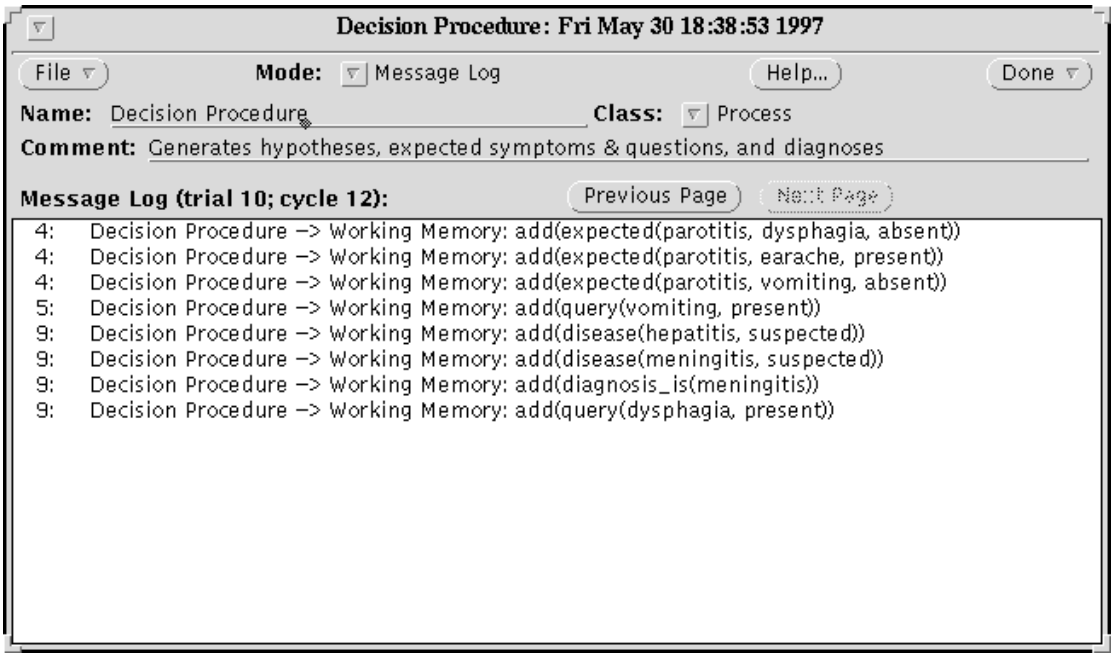

Figure 12: The message log of the Decision Procedure

2.5.4 Investigations of parameter spaces

A complaint frequently levelled at many computational models concerns their use of complex implementation details that go well beyond anything justified (and sometimes even justifiable) by the theoretical proposals which the models claim to reflect. Given that such implementation assumptions are often necessary for computational completeness, how can one determine which aspects of a model's behaviour can be correctly attributed to the underlying theory? One approach to this question is to investigate the effects of the so-called implementation assumptions on a model's behaviour.

Thus, if a theory posits a decaying buffer, without making a commitment to the precise form of that decay, we may investigate how behaviour varies as a function of different decay functions, possibly demonstrating that a range of decay functions are consonant with the empirical data. Alternatively, one may investigate the force of theoretical assumptions by investigating the extent to which behaviour is dependent upon those claims (cf. Cooper *et al.*, 1996). COGENT supports each of these approaches by allowing the investigation of dependencies between box configuration parameter settings and behaviour. Once a model is complete it is a simple matter to modify a parameter and run further trials to explore the criticality of the chosen parameter.

2.6 Miscellaneous box features

In addition to the features of components described above, all boxes have an associated description window in which free-form text describing the cognitive model it represents, its functioning, and its rationale, can be entered. Thus a typical box has five sorts of information associated with it (in addition to its name and class): its properties; its initial state; its current state; its message log; and its description.

Extensive printing facilities are also provided by the system. COGENT can generate PostScript files corresponding to complete research programmes, complete models (including box/arrow diagrams) or to individual boxes in a model. A variety of switches allows considerable control over the format of this output and the extent of information printed.

2.7 System documentation and help facilities

Each window within COGENT has a help button. This pops up a window giving details of the purpose, buttons and regions of the original window. A manual, which describes in detail all facilities and capabilities of the system, is available and a set of case studies is being compiled to accompany distribution of the system.

3. Concluding remarks

We have presented COGENT, a graphical environment for computational modelling that aims to simplify the process of developing cognitive models within a methodologically sound framework. In this final section we address some outstanding issues concerning coverage, system requirements, ongoing development, and distribution.

3.1 Existing models

To date, COGENT has been used to develop a number of models spanning several domains of cognitive psychology. This includes small scale models of concept combination (Cooper & Franks, 1996) and goal-based problem-solving (Cooper, 1996) to more extensive models such as performing the diagnosis task as discussed above (Fox & Cooper, in press) and learning the diagnosis task (Cooper & Fox, 1997). Other models include: child memory creation (Barreau, 1997); child memory recall (Miller, 1996); recall of future intentions (Ellis, Shallice & Cooper, in submission); effects of kinaesthetic training on children (Sims & Morton, in submission); “model-based” reasoning in tasks including syllogistic reasoning (Yule, 1997); and information retrieval in the complex domain of Human-Computer Interaction. We believe that the range of tasks and domains that have so far been modelled demonstrates the utility and flexibility of the system.

3.2 System requirements

Version 1 of COGENT was developed under UNIX/X-windows on a Sun workstation. It uses Sun’s Xview widget toolkit (which is freely available). In addition, a standard version of Prolog (such as SICStus Prolog or SWI Prolog) is required to run the system. We have ported the system to a variety of flavours of UNIX (including LINUX, SOLARIS 2.5, and SVR4) and have it running on several hardware platforms (Sun workstations, SGI workstations, and 486/Pentium PCs). As noted below, work is currently underway on porting the system to Microsoft Windows™ platforms.

3.3 Ongoing development

COGENT provides a rich and productive environment for computational modelling, but the list of possible features that such an environment might provide is almost endless. Development is therefore continuing. This development is driven partly by the demands of the existing user community and partly by our efforts to support sound modelling methodology. This section outlines the principal directions currently being pursued.

3.3.1 Support for "computational experiments"

The routine and rigorous methodology of experimental psychology is rarely matched within computational modelling. In order to address this weakness we are adding to COGENT tools to support the design, execution and analysis of computational experiments. At present it is possible to conduct simple experiments in COGENT by running a block of trials, but the present system does not explicitly support complex experimental designs involving multiple blocks with different stimulus sets and different parameter settings. (Such designs are possible, but they require significant intervention from the user throughout the experiment.) We are therefore developing a scripting language which will allow advanced users of the environment to specify and execute complex experimental designs.

Improved modelling tools, and in particular tools which enable rapid computational experiments, lead to large amounts of data. Whilst conventional tools can be used to present and analyse this data, we have found a clear need within COGENT for integrated display and analysis tools. We are therefore adding modules to display the results of computational experiments in tabular and graphical format. The final step along this route will be to incorporate standard statistical routines for the analysis of simulation data and the comparison of such data with human data.

3.3.2 Support for modelling neuropsychological damage

One of the areas of cognitive psychology where box/arrow modelling is currently most popular is cognitive neuropsychology. Current theorising here aims at producing box/arrow models which reflect functional modularity postulated to account for various deficits and dissociations. Computational modelling has also become an invaluable tool for investigating cognitive neuropsychological phenomena.[4] We are therefore developing tools to allow the selective lesioning of COGENT models. At present a model may be "damaged" by increasing decay on buffers or reducing the firing rates of rules. We are investigating a variety of further forms of damage (including, for example, limiting the transmission rates of arrows).

3.3.3 Support for hybrid modelling

COGENT is an environment primarily for symbolic modelling, but the inclusion of simple two-layered networks demonstrates that the underlying execution model is consistent with at least some of the requirements of connectionist modelling. A number of excellent systems for connectionist modelling are in existence, and we do not see COGENT as a competitor to these systems. However, hybrid symbolic/connectionist modelling has gained significant favour over the last 10 years, and we are not aware of any modelling packages or systems which support this enterprise. We are therefore attempting to develop suitably precise specifications for a variety of hybrid COGENT objects. Attention has so far focused on transducers, which map between symbolic/propositional representations and featural representations, and competitive networks, which use interactive activation between localist representations to resolve conflicts between multiple symbolic entities.

---

[4]Whilst most current modelling within cognitive neuropsychology is based on non-symbolic techniques (see, for example, Plaut & Shallice, 1994), there is no reason in principle why symbolic techniques (possibly involving probabilistic elements: see, for example, McCloskey, 1992) should be ruled out. Indeed, one view of connectionist neuropsychology is that it simply provides a way of operationalising box/arrow models by filling in the details of boxes. Given this, there is no *a priori* reason why a symbolic approach to such operationalising should be ruled out.

3.3.4 Porting to windows environments

As noted above, COGENT is currently only available on platforms running UNIX/X-windows. We are currently developing a version of the environment suitable for machines running varieties of Microsoft Windows™ (Windows 3.1, Windows 95, and Windows NT), and are considering porting the system to the Macintosh™ operating system. A final decision on this will depend on user demand.

3.4 Distribution policy

The X version of COGENT is currently available at no cost to academic sites who are willing to provide feedback on the system. The Microsoft Windows™ version will soon be available, for a small fee, to academic sites. A pricing structure for commercial sites is under development. Further information is available from:

`http://www.psyc.bbk.ac.uk/research/projects/cogent/`

4. References

Barreau, S. (1997): *Developmental Constraints on a Theory of Memory*. PhD Thesis. Department of Psychology, University College London.

Cooper, R. (1995): Towards an object oriented language for cognitive modelling. In J. D. Moore & J. F. Lehman (eds). *Proceedings of the 17th Annual Conference of the Cognitive Science Society*. Pittsburgh, PA. pp. 556–561.

Cooper, R. (1996): Perseverative subgoaling in production system models of problem solving. In G. W. Cottrell (ed). *Proceedings of the 18th Annual Conference of the Cognitive Science Society*. San Diego, CA. pp 396–402.

Cooper, R., Fox, J., Shallice, T., & Farringdon, J. (1996): A systematic methodology for cognitive modelling. *Artificial Intelligence, 85*, 3–44.

Cooper, R. & Fox, J. (1997): Learning to make decisions under uncertainty: The contribution of qualitative reasoning. To appear in *Proceedings of the 19th Annual Conference of the Cognitive Science Society*. San Francisco, CA.

Cooper, R. & Franks, B. (1996): The iteration of concept combination in sense generation. In G. W. Cottrell (ed). *Proceedings of the 18th Annual Conference of the Cognitive Science Society*. San Diego, CA. pp 523–528.

Ellis, J., Shallice, T., & Cooper, R. (in submission): Memory for, and the organisation of, future intentions.

Fox, J. & Cooper, R. (in press): Cognitive processing and knowledge representation in decision making. R. W. Scholz & A. C. Zimmer (eds.), *Qualitative Aspects of Decision Making*. Lengerich, Germany: Pabst Science Publishers.

Fox, J. (1980): Making decisions under the influence of memory. *Psychological Review, 87*. 190–211.

McCloskey, M. (1992). Cognitive mechanismsin numerical processing: Evidence from acquired dyscalculia. *Cognition, 44*. 107–157.

Miller, G. (1996): *A Headed Records Simulation of the Event Memory of Four Year Olds*. M.Sc. Thesis, Department of Computer Science, University College London.

Plaut, D. & Shallice, T. (1994): *Connectionist Modelling in Cognitive Neuropsychology*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Sims, K. & Morton, J. (in submission): Modelling the training effects of kinaesthetic acuity measurements in children.

Yule, P. (1997). Reasoning models in COGENT. Unpublished Manuscript. Department of Psychology, Birkbeck College, University of London.