Chapter 6

Models of the user in design

Overview

- User requirements modelling is concerned with establishing user needs.
  - Socio-technical models represent both human and technical requirements.
  - Soft systems methodology takes a broader view of human and organizational issues.
  - Participatory design incorporates the user directly into the design process.

- Cognitive models represent users of interactive systems.
  - Hierarchical models represent a user's task and goal structure.
  - Linguistic models represent the user–system grammar.
  - Physical and device models represent human motor skills.
  - Cognitive architectures underlie all of these cognitive models.

6.1 Introduction

In all engineering disciplines, the designer recruits a selection of models to contribute to the design process. If we were building a new office block, for example, then we would use models of air circulation to design the ventilation system, structural models for the fabric and possibly social models for the detailed design of the office layout.

Some models are *evaluative*, that is they tell us, after the fact, whether a given design has appropriate properties. For instance, we could use a structural analysis program to tell us whether a blueprint for a bridge will stand up to 40 tonne trucks driving over it. Other models are *generative*, that is they contribute during the design process itself, rather than merely commenting afterwards. In practice, all models are used to some extent in this generative manner. The engineer will have used less precise structural models in the design of the bridge before submitting it to the detailed analysis program. Also the program would be expected to say not only whether the bridge will fall down, but also where the weak spots are, thus contributing to the next stage of design.

Models are used in other disciplines too. We may analyze the structure of a piece of music and decide that it is a rondo, or say that a poem is in sonnet form. Further, we may deliberately set out to write a sonnet, thus imposing the model upon the creative process. Craft is the art of design within constraint, and models help to formulate the constraints.

This chapter and the next three describe a range of models that can be used during the interface design process. Just as in the design of the office block several different types of model are required for different aspects of the building, so in interface design we would expect to use a whole selection of complementary methods.

In this chapter we will look at two types of model. First we will consider the capture of user requirements within its social and organizational context. After this we will look at cognitive models which address aspects of users' perceptual and mental processes. Both types of model are highly user centred. The first looks outwards at the larger human context, the second is focused inwards at the individual user.

6.2 User requirements modelling

Requirements capture is an important part of all software engineering methodologies but often this activity focuses primarily on the functional requirements of the system – what the system must be able to do – with less emphasis on non-functional human issues such as usability and acceptability. Even where such matters are considered, they may reflect only the management's view of the user's needs rather than gathering information from the users themselves. User requirements modelling redresses this balance. There are a number of models and methods which can be used to capture a broader view of system requirements. In the next three sections we consider some of them, including socio-technical models, soft systems methodology, and participatory design.

6.3 Socio-technical models

Socio-technical models are concerned with technical, social, organizational and human aspects of design. They recognize the fact that technology is not developed in isolation but as part of a wider organizational environment. It is therefore important to consider social and technical issues side by side. There are a number of socio-technical models which are applied to the design of interactive computer systems but we will consider three: User Skills and Task Match (USTM) and its form for small organizations CUSTOM, Open System Task Analysis (OSTA), and Effective Technical and Human Implementation of Computer Systems (ETHICS).

6.3.1 USTM/CUSTOM

USTM is a socio-technical approach developed to allow design teams to understand and fully document user requirements [144]. It uses diagrammatic task models together with English descriptions to bring together structured methods and human factors. USTM has been customized for use in smaller organizations as CUSTOM [129], which focuses on establishing stakeholder requirements: all stakeholders are considered, not just the end-users. A stakeholder is defined as anyone who is effected by the success or failure of the system. Four categories of stakeholder are distinguished:

1. **Primary** those who use the system.
2. **Secondary** those who don't directly use the system but receive output from it or provide input to it (for example, someone who receives a report produced by the system).
3. **Tertiary** those who do not fall into 1 or 2 but who are affected by the success or failure of the system (for example, a director whose profits increase or decrease depending on the success of the system).
4. **Facilitating** those who are involved with design, development and maintenance of the system.

Example: Classifying stakeholders – an airline booking system
An international airline is considering introducing a new booking system for use by associated travel agents to sell flights directly to the public.

Primary stakeholders travel agency staff, airline booking staff

Secondary stakeholders customers, airline management

Tertiary stakeholders competitors, civil aviation authorities, customers' travelling companions, airline shareholders

Facilitating stakeholders design team, IT department staff.

CUSTOM is applied at the initial stage of design when a *product opportunity* has been identified. It is a forms-based methodology which provides a set of questions to apply at each of its stages:

1. Describe the organizational context, including its primary goals, physical characteristics, political and economic background.
2. Identify and describe stakeholders. All stakeholders are named, categorized (as primary, secondary, tertiary or facilitating) and described with regard to personal issues, their role in the organization and their job. For example, CUSTOM addresses issues such as stakeholder motivation, disincentives, knowledge, skills, power and influence within the organization, daily tasks and so on.
3. Identify and describe work-groups. A work-group is any group of people who work together on a task, whether formally constituted or not. Again, work-groups are described in terms of their role within the organization and their characteristics.
4. Identify and describe task–object pairs. These are the tasks that must be performed, coupled with the objects that are used to perform them or to which they are applied.
5. Identify stakeholder needs. Stages 2–4 are described in terms of both the current system and the proposed system. Stakeholder needs are identified by considering the differences between the two. For example, if a stakeholder is identified as lacking a particular skill currently which is required in the proposed system then a need for training is identified.
6. Consolidate and check stakeholder requirements. Here the stakeholder needs list is checked against the criteria determined at earlier stages.

A shorter version of CUSTOM stakeholder analysis

CUSTOM questions investigate a range of stakeholder characteristics, such as the following:

- ❑ What does the stakeholder have to achieve and how is success measured?
- ❑ What are the stakeholder's sources of job satisfaction? What are the sources of dissatisfaction and stress?
- ❑ What knowledge and skills does the stakeholder have?
- ❑ What is the stakeholder's attitude towards work and computer technology?
- ❑ Are there any work-group attributes that will affect the acceptability of the product to the stakeholder?
- ❑ What are the characteristics of the stakeholder's task in terms of frequency, fragmentation and choice of actions?
- ❑ Does the stakeholder have to consider any particular issues relating to responsibility, security or privacy?
- ❑ What are the physical conditions in which the stakeholder is working?

CUSTOM provides a useful framework for considering stakeholder requirements and the use of forms and questions (a 'manual' for its use is available) makes it relatively straightforward, if somewhat time consuming, to apply. However, in less complex situations, it is possible to use a shortened version of CUSTOM stakeholder analysis (see Box on page 225) for stages 2–4.

6.3.2 OSTA

OSTA [74] attempts to describe what happens when a technical system is introduced into an organizational work environment. In OSTA, the social aspects of the system (such as usability and acceptability) are specified together with the technical aspects (such as system functionality).

OSTA has eight main stages:

1. The primary task which the technology must support is identified in terms of users' goals.
2. Task inputs to the system are identified. These may have different sources and forms which may constrain the design.
3. The external environment into which the system will be introduced is described, including physical, economic and political aspects.
4. The transformation processes within the system are described in terms of actions performed on or with objects.
5. The social system is analyzed, considering existing work-groups and relationships within and external to the organization.
6. The technical system is described in terms of its configuration and integration with other systems.
7. Performance satisfaction criteria are established, indicating the social and technical requirements of the system.
8. The new technical system is specified.

The outcomes of OSTA are presented using notations familiar to designers such as data flow diagrams and textual descriptions.

6.3.3 ETHICS

ETHICS [165] is also concerned with establishing social and technical requirements but differs from OSTA in that it addresses the two parallel strands of design – the social and the technical – using different design teams. In the ETHICS method, the design teams work separately and then attempt to merge their solutions to find the most effective solution which is compatible with both the social and technical requirements that have been identified.

There are six key stages in ETHICS:

1. The problem is identified and the current system described. Objectives and tasks are identified, as are information needs and job satisfaction requirements. Constraints on the system, both technical and social, are identified.

2. Two design teams are established, one to examine social aspects, the other technical. The objectives and needs identified in stage 1 are ranked in order of priority and are checked for compatibility before social and technical design decisions are made.
3. Alternative social and technical solutions are set out and evaluated against the criteria already established to determine a short list of possibilities.
4. Solutions from stage 3 are checked for compatibility.
5. Compatible pairs of socio-technical solutions are ranked according to the criteria already agreed.
6. Detailed designs are developed.

The ETHICS approach attempts to reach a solution which meets both user and task requirements by having specialist teams rank potential solutions and selecting the one which ranks highly on both social and technical criteria. The emphasis is on reaching a solution which ranks highly on job satisfaction to ensure that the solution is acceptable.

6.4 Soft systems methodology

The socio-technical models we have looked at focus on identifying requirements from both human and technical perspectives. Soft systems methodology (SSM) takes an even broader view: that of the organization as a system of which the technology and people are components. SSM was developed by Checkland [46] to help designers reach an understanding of the context of technological developments: the emphasis is therefore on understanding the situation rather than on devising a solution. SSM has seven stages. A distinction is made between the 'real world' stages (1–2, 5–7) and the systems stages (3–4).

The first stage of SSM is the recognition of the problem and initiation of analysis. This is followed by a detailed description of the problem situation: developing a *rich picture*. This will include all the stakeholders, the tasks they carry out and the groups they work in, the organizational structure and its processes and issues raised by any stakeholder. Any knowledge elicitation techniques can be used to gather the information to build the rich picture, including observation (and video and audio recording), structured and unstructured interviews and questionnaires, and workshops incorporating such activities as role play, simulations and critical incident analysis. In general the less structured approaches should be used initially to avoid artificially constraining the description. The rich picture can be in any style – there are no right or wrong answers – but it should be clear and informative to the designer.

At the next stage we move from the real world to the systems world and attempt to generate *root definitions* for the system. There may be several root definitions of a system, representing different stakeholder perspectives for example. Root definitions are described in terms of CATWOE:

Clients those who receive output or benefit from the system.

Actors those who perform activities within the system.

Transformations the changes that are affected by the system. This is a critical part of the root definition as it leads to the activities which need to be included in the next stage. To identify the transformations, consider the inputs and outputs of the system.

Weltanschauung (from the German) or world view. This is how the system is perceived in a particular root definition.

Owner those to whom the system belongs, to whom it is answerable and who can authorize changes to it.

Environment the world in which the system operates and by which it is influenced.

Root definition for airline management: airline booking system
An international airline is considering introducing a new booking system for use by associated travel agents to sell flights directly to the public. That is, a system owned by the airline management, operated by associated travel agency staff, working in associated travel agency offices worldwide, operating within regulations specified by international civil aviation authorities and national contract legislation, to sell flights to and reserve seats for customers and to generate a profit for the company.

Client Customer.

Actor Travel agency staff.

Transformation Customer's intention and request to travel transformed into sale of seat on flight and profit for organization.

Weltanschauung Profits can be optimized by more efficient sales.

Owner Airline management.

Environment Regulations of international civil aviation authorities and national contract legislation. Local agency policies worldwide.

Once the root definitions have been developed, the *conceptual model* is devised. The conceptual model defines what the system has to do to fulfil the root definitions. It includes identifying the transformations and activities in the system and modelling them hierarchically in terms of what is achieved and how it is achieved. This process is iterative and is likely to take several cycles before it is complete and accurate.

Next we return to the real world with our systems descriptions and compare the actual system with the conceptual model, identifying discrepancies and thereby highlighting any necessary changes or potential problems. For example, a particular activity may have more processes in the real world than in the conceptual model which may suggest that a reduction of processes for that activity is needed.

In the final stages we determine which changes are necessary and beneficial to the system as a whole – changes may be structural, procedural or social, for example – and decide on the actions required to affect those changes.

SSM is a flexible approach which supports detailed consideration of the design context. However, it takes practice to use effectively. There is no single right (or wrong) answer – the SSM is successful if it aids the designer's understanding of the wider system.

6.5 Participatory design

Participatory design is a philosophy which encompasses the whole design cycle. Participatory design is design in the workplace, incorporating the user not only as an experimental subject but as a member of the design team. Users are therefore active collaborators in the design process, rather than passive participants whose involvement is entirely governed by the designer. The argument is that users are experts in the work context and a design can only be effective within that context if these experts are allowed to contribute actively to the design. In addition, introduction of a new system is liable to change the work context and organizational processes, and will only be accepted if these changes are acceptable to the user. Participatory design therefore aims to refine system requirements iteratively through a design process in which the user is actively involved.

Participatory design has three specific characteristics. It aims to improve the work environment and task by the introduction of the design. This makes design and evaluation context or work oriented rather than system oriented. Secondly, it is characterized by collaboration: the user is included in the design team and can contribute to every stage of the design. Finally, the approach is iterative: the design is subject to evaluation and revision at each stage.

Participatory design originated in Scandinavia, where it is now promoted in law and accepted work practices. Although principles have been adopted from the approach elsewhere, it has not been widely practised. This may be due to the time and cost involved in what is, by definition, a context-specific design, as well as the organizational implications of the shift of power and responsibility.

The participatory design process utilizes a range of methods to help convey information between the user and designer. They include

brainstorming This involves all participants in the design pooling ideas. This is informal and relatively unstructured although the process tends to involve 'on-the-fly' structuring of the ideas as they materialize. All information is recorded without judgement. The session provides a range of ideas from which to work. These can be filtered using other techniques.

storyboarding This has been discussed in more detail in Chapter 5. Storyboards can be used as a means of describing the user's day-to-day activities as well as the potential designs and the impact they will have.

workshops These can be used to fill in the missing knowledge of both participants and provide a more focused view of the design. They may involve mutual enquiry in which both parties attempt to understand the context of the design from each other's point of view. The designer questions the user about the work environment in which the design is to be used, and the user can query the designer on the technology and capabilities that may be available. This establishes common ground between the user and designer and sets the foundation for the design that is to be produced. The use of role play exercises can also allow both user and designer to step briefly into one another's shoes.

pencil and paper exercises These allow designs to be talked through and evaluated with very little commitment in terms of resources. Users can 'walk through' typical tasks using paper mock-ups of the system design. This is intended to show up discrepancies between the user's requirements and the actual design as proposed. Such exercises provide a simple and cheap technique for early assessment of models.

These methods are obviously not exclusively used in participatory design. They can be used more widely to promote clearer understanding between designer and user. Often the design context (for example, the constraints of a particular organization) does not permit full-blown participatory design. Even if this is the case, methods such as these are useful ways of encouraging cooperation between the two parties.

6.6 Cognitive models

The remaining techniques and models in this chapter all claim to have some representation of users as they interact with an interface; that is, they model some aspect of the user's understanding, knowledge, intentions or processing. The level of representation differs from technique to technique – from models of high-level goals and the results of problem-solving activities, to descriptions of motor-level activity, such as keystrokes and mouse clicks. The formalisms have largely been developed by psychologists, or computer scientists whose interest is in understanding user behaviour.

One way to classify them is in respect to how well they describe features of the *competence* and *performance* of the user. Quoting from Simon [221]:

> Competence models tend to be ones that can predict legal behaviour sequences but generally do this without reference to whether they could actually be executed by users. In contrast, performance models not only describe what the necessary behaviour sequences are but usually describe both what the user needs to know and how this is employed in actual task execution.

Competence models, therefore, represent the kinds of behaviour expected of a user, but they provide little help in analyzing that behaviour to determine its demands on

the user. Performance models provide analytical power mainly by focusing on routine behaviour in very limited applications.

Another useful distinction between these models is whether they address the acquisition or formulation of a plan of activity or the execution of that plan. Referring back to the interaction framework presented in Chapter 3, this classification would mean that some models are concerned with understanding the *User* and his associated task language while others are concerned with the articulation translation between that task language and the *Input* language. The presentation of the cognitive models in this chapter follows this classification scheme, divided into the following categories:

- hierarchical representation of the user's task and goal structure.
- linguistic and grammatical models
- physical and device-level models.

The first category deals directly with the issue of formulation of goals and tasks. The second category deals with the grammar of the articulation translation and how it is understood by the user. The third category again deals with articulation, but at the human motor level instead of at a higher level of human understanding.

Architectural assumptions about the user are needed in any of the cognitive models discussed here. Some of the more basic architectural assumptions were covered in Chapter 1, such as the distinction between long- and short-term memory. After discussing models in the three categories above, we will describe two additional cognitive architectures and how they are relevant for analyzing interactive system design.

Many of these nominally cognitive models have a rather computational flavour. This reflects the way that computational analogies are currently used in cognitive psychology. The similarity between the language describing the user and that describing the computer has some advantages and some dangers. On the positive side it makes communication and analysis of the combined human–computer system easier. For instance, cognitive complexity theory (described later) produces models of both user goals and the system grammar, and can reason about their interaction. On the other hand, there is a danger that this will encourage a mechanistic view of the user.

6.7 Goal and task hierarchies

Many models make use of a model of mental processing in which the user achieves goals by solving subgoals in a divide-and-conquer fashion. We will consider two models, *GOMS* and *CCT*, where this is a central feature. However, we will see similar features in other models, such as *TAG* (Section 6.8.2) and when we consider task analysis techniques (Chapter 7).

Imagine we want to produce a report on sales of introductory HCI textbooks. To achieve this goal we divide it into several subgoals, say gathering the data together,

producing the tables and histograms, and writing the descriptive material. Concentrating on the data gathering, we decide to split this into further subgoals: find the names of all introductory HCI textbooks and then search the book sales database for these books. Similarly each of the other subgoals is divided up into further subgoals, until some level of detail is found at which we decide to stop. We thus end up with a hierarchy of goals and subgoals. The example can be laid out to expose this structure:

```
produce report
   gather data
   .   find book names
   .   .   do keywords search of names database
                  << further subgoals >>
   .   .   sift through names and abstracts by hand
                  << further subgoals >>
   .   search sales database
              << further subgoals >>
   layout tables and histograms
       << further subgoals >>
   write description
       << further subgoals >>
```

Various issues arise as one attempts such analyses of computer use.

Where do we stop? We can go on decomposing tasks until we get down to the individual hand and eye movements of the user, or we can stop at a more abstract level. Where do we start? In a similar way, we can start our analyses at different points in the hierarchy of goals. At the extreme we could extend our analysis to larger and larger goals: 'light oven' is a subgoal of 'boil peas' and so on to goals such as 'have my dinner', 'feed' and 'stay alive'.

These two questions are issues of *granularity*, and both of the methods described below leave this to some extent in the hands of the designer. Different design issues demand different levels of analysis. However, both methods operate at a relatively low level; neither would attempt to start with such an abstract goal as 'produce a report' which will involve real creativity and difficult problem solving. Instead they confine themselves to more routine learned behaviour. This most abstract task is referred to as the *unit task*. The unit task does not require any problem-solving skills on the part of the user, though it frequently demands quite sophisticated problem-solving skills on the part of the designer to determine them.

What do we do when there are several ways of solving a problem, or if the solutions to two subgoals interact? Users will often have more than one way to achieve a goal and there must be some way of representing how they select between competing solutions.

Another important issue has to do with the treatment of error. Users are not perfect. A goal hierarchy may show how the perfect user would achieve a goal, but what can it say about difficulties the user may have along the way? In general, prediction of error behaviour is poor amongst these hierarchical modelling techniques, though some (cognitive complexity theory (CCT), for example) can represent error behaviour.

6.7.1 GOMS

The *GOMS* model of Card, Moran and Newell is an acronym for Goals, Operators, Methods and Selection [37]. A GOMS description consists of these four elements:

Goals These are the user's goals, describing what the user wants to achieve. Further, in GOMS the goals are taken to represent a 'memory point' for the user, from which he can evaluate what should be done and to which he may return should any errors occur.

Operators These are the lowest level of analysis. They are the basic actions that the user must perform in order to use the system. They may affect the system (for example, press the 'X' key) or only the user's mental state (for example, read the dialog box). There is still a degree of flexibility about the granularity of operators; we may take the command level 'issue the SELECT command' or be more primitive: 'move mouse to menu bar, press centre mouse button ...'.

Methods As we have already noted, there are typically several ways in which a goal can be split into subgoals. For instance, in a certain window manager a currently selected window can be closed to an icon either by selecting the 'CLOSE' option from a pop-up menu, or by hitting the 'L7' function key. In GOMS these two goal decompositions are referred to as methods, so we have the `CLOSE-METHOD` and the `L7-METHOD`:

```
GOAL: ICONIZE-WINDOW
.   [select GOAL: USE-CLOSE-METHOD
.           .   MOVE-MOUSE-TO-WINDOW-HEADER
.           .   POP-UP-MENU
.           .   CLICK-OVER-CLOSE-OPTION
.           GOAL: USE-L7-METHOD
.           .   PRESS-L7-KEY]
```

The dots are used to indicate the hierarchical level of goals.

Selection From the above snippet we see the use of the word `select` where the choice of methods arises. GOMS does not leave this as a random choice, but attempts to predict which methods will be used. This typically depends both on the particular user and on the state of the system and details about the goals. For instance, a user, Sam, never uses the `L7-METHOD`, except for one game, '`blocks`', where the mouse needs to be used in the game until the very moment the key is pressed. GOMS captures this in a selection rule for Sam:

User Sam:
`Rule 1:` Use the `CLOSE-METHOD` unless another rule applies.
`Rule 2:` If the application is '`blocks`' use the `L7-METHOD`.

The goal hierarchies described in a GOMS analysis are almost wholly below the level of the unit task defined earlier. A typical GOMS analysis would therefore consist of a single high-level goal which is then decomposed into a sequence of unit tasks, all of which can be further decomposed down to the level of basic operators:

GOAL: EDIT-MANUSCRIPT
. GOAL: EDIT-UNIT-TASK *repeat until no more unit tasks*

The goal decomposition between the overall task and the unit tasks would involve detailed understanding of the user's problem-solving strategies and of the application domain. These are side-stepped entirely by the method as originally proposed. It would be possible to use the general notation in order to describe this subgoal structure (as for instance in the book report example above). This form of high-level goal description is adopted during *task analysis* which will be discussed in Chapter 7. In particular, the aim of *hierarchical task analysis* is to produce task decompositions, which would be similar (but in a different notation) to that in the book report example.

Analysis of the GOMS goal structure can yield measures of performance. The stacking depth of a goal structure can be used to estimate short-term memory requirements. The model of the users' mental processes implied by this is, of course, very idealized. Also, the selection rules can be tested for accuracy against user traces, and changed in response to discrepancies. In early experiments on the technique, the inventors were able to achieve on average 90% correct prediction rate of user commands. Further, a very simple method of predicting times (basically assuming that each operator takes a constant time) was able to predict actual times with an error of 33%.

The original GOMS model has served as the basis for much of the cognitive modelling research in HCI. It was good for describing how experts perform routine tasks. Coupled with the physical device models discussed later, it can be used to predict the performance of these users in terms of execution times. It was never intended to provide the kind of information about the user's knowledge that could be compared across different tasks in order to predict things like training or transfer times.

Design Focus

GOMS saves money

A few years ago the US telephone company NYNEX were intending to install a new computer system to support their operators. Before installation a detailed GOMS analysis was performed taking into account the cognitive and physical processes involved in dealing with a call. The particular technique was rather different from the original GOMS notation as described here. Because an operator performs several activities in parallel a PERT-style GOMS description was constructed [123, 101]. The PERT analysis was used to determine the critical path, and hence the time to complete a typical task. It was discovered that rather than speeding up operations, the new system would take longer to process each call. The new system was abandoned before installation, leading to a saving of many millions of dollars.

Worked exercise

Create a GOMS description of the task of photocopying an article from a journal. Discuss the issue of closure in terms of your GOMS description.

Answer

One possible GOMS description of the goal hierarchy for this task is given below. Answers will vary depending on assumptions about the photocopier used as the model for the exercise. In this example, we will assume that the article is to be copied one page at a time and that a cover over the imaging surface of the copier has to be in place before the actual copy can be made.

```
GOAL: PHOTOCOPY-PAPER
.     GOAL: LOCATE-ARTICLE
.     GOAL: PHOTOCOPY-PAGE repeat until no more pages
.     .     GOAL: ORIENT-PAGE
.     .     .     OPEN-COVER
.     .     .     SELECT-PAGE
.     .     .     POSITION-PAGE
.     .     .     CLOSE-COVER
.     .     GOAL: VERIFY-COPY
.     .     .     LOCATE-OUT-TRAY
.     .     .     EXAMINE-COPY
.     GOAL: COLLECT-COPY
.     .     LOCATE-OUT-TRAY
.     .     REMOVE-COPY  (outer goal satisfied!)
.     GOAL: RETRIEVE-JOURNAL
.     .     OPEN-COVER
.     .     REMOVE-JOURNAL
.     .     CLOSE-COVER
```

The closure problem which appears in this example occurs when the copy of the article is removed from the photocopier out tray, satisfying the overall goal for the task. In the above description, however, the original journal article is still on the imaging surface of the photocopier, and the cover is closed. The user could easily forget to remove the journal. How could the photocopying procedure be revised to eliminate this problem? One answer is to force the goal `RETRIEVE-JOURNAL` to be satisfied before `COLLECT-COPY`.

6.7.2 Cognitive complexity theory

Cognitive complexity theory (CCT), introduced by Kieras and Polson [128], begins with the basic premises of goal decomposition from GOMS and enriches the model to provide more predictive power. CCT has two parallel descriptions: one of the user's goals and the other of the computer system (called the *device* in CCT). The description of the user's goals is based on a GOMS-like goal hierarchy, but is expressed primarily using *production rules*. We introduced production rules in Chapter 1 and we further describe their use in CCT below. For the system grammar, CCT uses *generalized transition networks*, a form of *state transition network*. This

will not be described here, but state transition networks will be discussed in detail in Chapter 8.

The production rules are a sequence of rules:

if *condition* then *action*

where *condition* is a statement about the contents of working memory. If the condition is true then the production rule is said to fire. An *action* may consist of one or more elementary actions, which may be either changes to the working memory, or external actions such as keystrokes. The production rule 'program' is written in a LISP-like language.

As an example, we consider an editing task using the UNIX **vi** editor. The task is to insert a space where one has been missed out in the text, for instance if we noticed that in the above paragraph we had written 'cognitivecomplexity theory'. This is a reasonably frequent typing error and so we assume that we have developed good procedures to perform the task. We consider a fragment of the associated CCT production rules.

```
(SELECT-INSERT-SPACE
IF (AND (TEST-GOAL perform unit task)
        (TEST-TEXT task is insert space)
        (NOT (TEST-GOAL insert space))
        (NOT (TEST-NOTE executing insert space)) )
THEN (  (ADD-GOAL insert space)
        (ADD-NOTE executing insert space)
        (LOOK-TEXT task is at %LINE %COL) ))

(INSERT-SPACE-DONE
IF (AND (TEST-GOAL perform unit task)
        (TEST-NOTE executing insert space)
        (NOT (TEST-GOAL insert space)) )
THEN (  (DELETE-NOTE executing insert space)
        (DELETE-GOAL perform unit task)
        (UNBIND %LINE %COL) ))

(INSERT-SPACE-1
IF (AND (TEST-GOAL insert space)
        (NOT (TEST-GOAL move cursor))
        (NOT (TEST-CURSOR %LINE %COL)) )
THEN (  (ADD-GOAL move cursor to %LINE %COL) ))

(INSERT-SPACE-2
IF (AND (TEST-GOAL insert space)
        (TEST-CURSOR %LINE %COL) )
THEN (  (DO-KEYSTROKE 'I')
        (DO-KEYSTROKE SPACE)
        (DO-KEYSTROKE ESC)
        (DELETE-GOAL insert space) ))
```