Chapter 4

Cognitive Models and Architectures

In chapter 2, we saw that different collaborative groups can be characterized according to the different types of information they use and produce over the course of a project and the flow of information they generate in transforming one type into another. In chapter 3, several kinds of computer and communication tools were discussed that support these information processing activities. In this chapter, I look at models and architectures that describe human cognition as a form of information processing system. These three perspectives, then, provide the basic materials needed to build a concept of collective intelligence as a form of computer-mediated behavior in which human beings supply the mental processes used to build large, complex structures of ideas.

The chapter is divided into three sections. In the first, I review general cognitive models and architectures. Next, I look at specialized models or frameworks that apply these ideas to particular tasks or situations encountered by collaborative groups. In the third section, I look briefly at an objection raised by Allen Newell to the idea of group intelligence. The goal of the first two sections is to identify a basic set of components, broader than any single theory, that have been used to describe human mental function. In Part II, I identify constructs within computer-based collaborative groups that are recognizable as extrapolations of these components as a starting point for building a concept of collective intelligence. I also try to answer Newell's objection.

General IPS Models and Architectures

For the past 20 years, cognitive science has been dominated by the view that human cognition is fundamentally concerned with the processing of information. In this section, I review Newell's and

Simon's original Information Processing System (IPS) model that established this perspective. After that, I look at two more recent models that sum-up much of the intervening work. These include Newell's IPS-based cognitive architecture and an alternative architecture developed by John Anderson.

IPS

The original Newell and Simon model explains how human beings carry out complex problem-solving tasks (Newell & Simon, 1972). Their goal was not to develop an abstract cognitive model as an end in itself but, rather, to develop computer programs that could simulate human intelligence. They based their model on think-aloud protocols of human subjects solving three types of problems — cryptarithmetic, theorem proving, and chess — but they also incorporated a number of other results from the literature into the model.

A key assumption in their work was that cognition is inherently concerned with processing information. Hence, their model emphasizes the representation, processing, and storage of information and has come to be called the Information Processing System perspective. A high-level view of this model is shown in Fig. 4.1. Its major components include *Receptor* and *Effector* functions that interact with the external environment, a *Memory*, and a *Processor* that operates on information flowing among these components.

Because the entire system is oriented toward the flow and control of information, a key concept is *symbol*, the primitive unit of information on which and by which the Information Processing System works. Although the model suggests that effector and receptor functions map between external phenomena and symbols internal to the system, these components are not developed in any detail in the model. Rather, information is directly inserted into the system or obtained from it by the researcher. Thus, the model is most concerned with the processing of information after it already exists in symbolic form within the system.

Although symbols can be stored individually in memory, they are normally organized into *symbol structures*. These structures may be *designative*, in which case they denote semantic information, or they may be *programs*, in which case they represent operations or *methods* that can be applied to symbol structures to derive information about

them — for example, compare two structures for equivalence — or to change them — for example, add a new symbol to the structure. Designative information is stored in the memory as associative structures of symbols organized into hierarchical chunks. Methods, on the other hand, are stored as production rules.

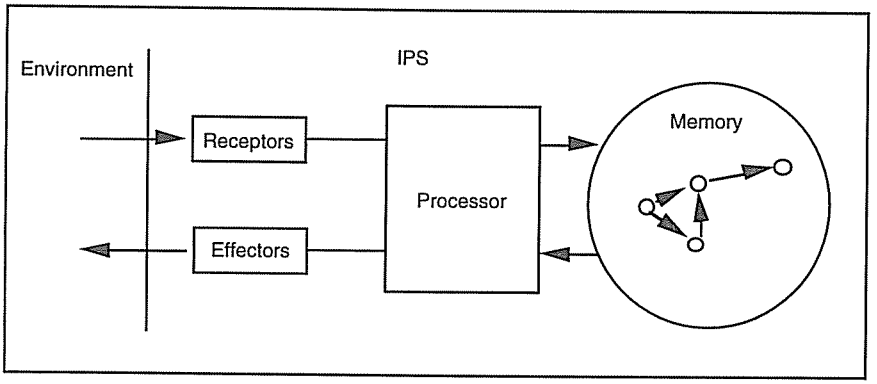

Fig. 4.1. Newell and Simon's Information Processing System (IPS) model of cognition. (Adapted by permission from Newell & Simon, 1972.)

The processor provides the context in which elementary cognitive processes, represented as production rules, operate on the semantic contents of memory, represented as a graph structure. Not shown in the figure is a short-term memory (STM) component that is part of the processor. It provides a cache that is loaded from memory as well as from perceptual mechanisms that are part of the receptor. Processes operate on the symbols/symbol structures within it; subsequently, its contents can be encoded and stored in the memory.

The IPS model becomes animated through the process of solving problems. A problem is presented to the system with respect to an external *task environment.* However, the system does not operate on the problem within that external context but, rather, within an interior *problem space.* The problem space contains a goal — the solution to the problem — a representation of the problem, a representation of relevant aspects of the task environment, and a data structure consistent with these representations. Methods are applied to the initial representation of the problem, producing incremental changes in the data structure. The process is repeated until a path is

constructed that joins the initial problem to the data state that represents the goal. However, if along the way the processor reaches an impasse and cannot complete the construction, it generates one or more subgoals in an attempt to resolve the impasse. Each new subgoal, in turn, generates a new problem space. Frequently, the new problem space requires translating the problem or subproblem into an alternative representation appropriate for that context, and it is often this change in representation that enables a solution to the problem. Once the subgoal is achieved, processing returns to the higher level problem space in which the subgoal was generated and continues from there. This hierarchical nesting is both recursive and iterative and, in theory, unlimited in the number of levels of descent that can be generated.

From this brief description, we can identify several basic components found in the IPS model. These include the memory, the processor, and its short-term memory component. Effector and receptor functions are also included, but they are not developed in any detail in the model. At a more abstract level, the problem-solving system also includes goals, problem spaces, and their internal representations and data structures. Although these latter constructs play key roles in human problem-solving activities, Newell and Simon regarded them as task-dependent and, thus, informational objects learned by the individual problem-solver rather than as basic components of the IPS model.

Soar

In the 20 years since the publication of the original IPS model, Newell's views of human cognition obviously changed. But what is most striking is not the differences in those views but their continuity. The changes represent extrapolations and refinements, rather than completely different formulations. Newell's more recent goal was to develop a comprehensive cognitive architecture that could provide a framework in which to develop unified theories of cognition — systems that incorporate the sum of what is known about human cognition (Newell, 1990). The vehicle for supporting such theories Newell called *Soar*.

Soar is several things at once. First, Newell emphasized that Soar is an architecture, rather than a model. By that he meant that Soar includes as basic constituents those mental functions and constructs that remain fixed under different tasks and conditions. For example, the mechanism that performs memory accesses is regarded as invariant and, thus, architectural, whereas the contents of a particular memory access is variable and, thus, data. This separation between general and specific makes it possible to define different cognitive models within the general Soar framework. Second, Soar is a computer programming language in which to write Artificial Intelligence (AI) programs. It is based on the OPS-5 expert system language but includes several enhancements, including a problem space construct and chunking mechanism that are basic parts of the Soar language. Third, Soar also refers to individual computer programs, written in the Soar language, that simulate specific cognitive tasks and processes or implement specific cognitive models. Soar programs have been written that duplicate the original problem-solving tasks described in Newell and Simon (1972), perform syllogistic reasoning, verify elementary sentences, acquire skills through experience, and simulate an impressive list of other cognitive functions.

Although Soar bears a strong resemblance to the IPS model, it is also different in several important ways. In the IPS model memory and processor were separate components, and the processor contained within it a small working cache, called the short-term memory. In the Soar architecture, shown in Fig. 4.2, memory is divided into two components — long-term memory and working memory. Furthermore, the processor has disappeared as a separate component and is replaced by a set of basic cognitive processes, a more abstract concept. These processes are stored in long-term memory but operate within the new working memory component. Another major distinction is that the associative network structure of long-term memory in the 1972 model has been replaced in the Soar architecture by additional production rules. Thus, both declarative knowledge as well as cognitive processes are represented as productions.

A Soar system becomes animated through a process of *searching*, similar to that in the IPS model. Operators or methods are applied to a data structure in order to achieve a goal state within a problem space. Thus, all processing takes place within the context of the particular problem space that is currently loaded into the working memory component of the system. The problem space construct is similar to that for the IPS, but in Soar it is regarded as a fundamental

part of the architecture that applies to all human beings rather than as an ad hoc information structure learned by an individual. Thus, all processing is cast within the general paradigm of constructing a path from the initial problem state to the goal state and of resolving the impasses that arise along the way. However, a major difference is the way Soar learns from resolving impasses. Once an impasse is resolved, Soar engages a process called *chunking*. A new production rule is created that records, as the conditional part of the rule, the state of the problem space at the time the impasse was encountered and, as the action portion of the rule, the path to solution. Chunking is Soar's primary learning mechanism and is a basic part of the Soar architecture, rather than an ad hoc procedure.

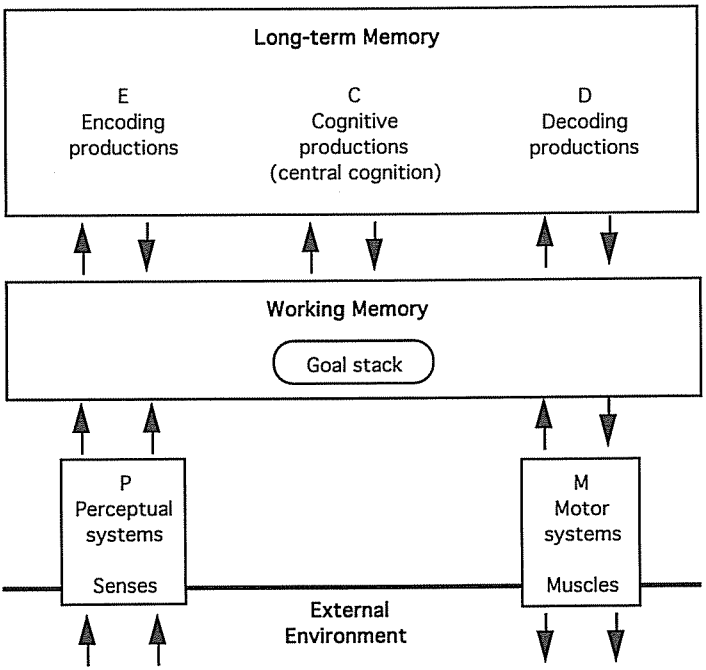

Fig. 4.2. Newell's Soar cognitive architecture. (Adapted by permission from Newell, 1990.)

Thus, Soar has retained the IPS's long-term memory component and mechanisms for interacting with the external world, although these latter components are still undeveloped as they were in IPS.

However, important changes have been made in the structure of long-term memory and in the disappearance of the processor component. Both semantic and procedural knowledge are stored as production rules in Soar. The processor has been replaced by a working memory component that provides the context in which processing takes place. Although particular problem spaces are task specific and, hence, informational, the underlying form common to all problem spaces is invariant. Consequently, both problem spaces and the primitive functions that operate on them — to select a particular problem space, to select a data state within it, and to select and apply operators to that state — are considered to be basic parts of the architecture. Finally, the impasse resolution and chunking functions provide the principle mechanisms for learning and give Soar much of its generality and flexibility. They, too, are regarded as part of the architecture.

Act*

A second comprehensive architecture has been developed by Newell's long-time CMU colleague, John Anderson (Anderson, 1983, 1990). Newell called Anderson's Act* the first unified theory of cognition (Newell, 1990). It grew out of Anderson's earlier work on the semantic structure of human memory, modeled as an associative network (Anderson & Bower, 1973). Since then, Anderson has added a process component, in the form of production rules, that acts on the contents of memory and a skills acquisition feature that allows rules to perform actions on other rules. Like Soar, Act* is implemented as a computer system. Act* programs can simulate a wide spectrum of mental behaviors ranging from basic cognitive processes, such as fact retrieval and stimulus-response behaviors, to higher level functions and problem-solving tasks, such as language acquisition and constructing geometric proofs.

The Act* architecture, shown in Fig. 4.3, is similar to Newell's Soar system in its interactions with the external environment and in its distinction between long-term memory and working memory. But it differs from Newell's system in several important respects. Among these differences are its division of long-term memory into two separate storage systems, its incorporation of activation as a fundamental part of the architecture, and its inclusion of multiple data types for individual node contents within the declarative memory.

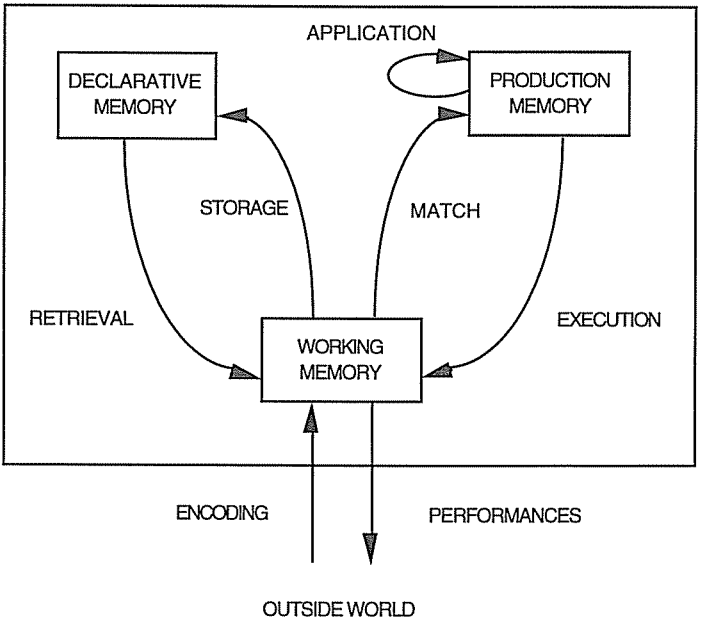

Fig. 4.3. Anderson's Act* cognitive architecture. (Adapted by permission from Anderson, 1983.)

Anderson divided long-term memory into two separate stores, which he called *production memory* and *declarative memory*. Declarative memory is the store for facts and is structured as a semantic network. Nodes in the graph store information as short sequences, images, and propositions; more complex structures are composed of hierarchies of these basic lower level types. Thus, there are different storage subsystems in the declarative memory for different types of information. Nodes are joined to one another through associative relations and each has a level of activation. Production memory is the store for processes, represented as production rules. These rules operate on the contents of declarative memory, but they also operate on the contents of the production memory itself to produce new rules, modify existing rules, and so on.

Working memory is the context in which processes are applied to the contents of the declarative and production memories. Thus, it functions as a form of problem space, but unlike Newell's systems that

include multiple problem spaces, it is the sole problem space in Anderson's architecture. The figure is misleading with respect to working memory in one important respect. Working memory is not a separate component in Anderson's system but, rather, that portion of the other two memory systems that is currently activated. A process of spreading activation determines which parts of the two memory systems are the working memory at any given moment. As processing takes place and, thus, activation levels and conditions for rule selection change, so working memory changes. By analogy, think of looking down on a darkened surface — the long-term memory systems — and sweeping a flashlight across that surface, selectively illuminating different parts — activating or condition selecting different parts of the memory stores. The currently lighted/activated/condition-selected parts constitute the working memory at that moment. This concept of working memory stands in sharp contrast with the view that it is a separate component into which contents are copied from long-term memory and vice versa.

Since the original description of the Act* architecture in 1983, Anderson extended the capability of the system so that it can formulate more sophisticated strategies and adapt its behavior to changes in the environment (Anderson, 1990). This is done through a three-step process in which the goals of the system are first determined, then a formal model of the environment is developed, and, third, the computational limits of the system are identified. From these three sources of information, an *optimal behavior function* is defined for the system in its current context relative to current goals that is then applied to the relevant problem or task. Because the process may produce errors, it can be iterated until the system produces behaviors that are consistent with empirical results. Thus, the additional level adds a form of *rationality* to the Act* architecture.

In summary, Anderson's Act* system grew out of his earlier work on human memory. It is similar to Newell's Soar system in several respects. It represents cognitive processes as production rules. It employs a type of problem space construct as the context for higher level functions. And it controls its behavior through a hierarchical goal-subgoal structure. However, Act* differs from Soar in several ways. It includes two separate memory systems — a production memory for processes and a declarative memory for semantic information. The declarative memory is organized as a network of nodes whose contents are represented in terms of a small set of basic

data types. Activation plays an important role in Act*. Nodes and productions have associated activation values and a spreading activation process figures prominently in the architecture. Thus, its working memory is not a separate component but the currently activated portion of its memory systems. Although Anderson's recent work in rational analysis may lead to a more sophisticated concept of strategy, that remains a weakness in the Act* architecture as it is for other IPS systems. Act* is similarly limited with respect to perception and motor actions.

From these three IPS models/architectures, we can sketch a composite view of cognitive systems and their components. If we wish to regard collective intelligence as an IPS system, it, too, is likely to have components similar to these or components that are at least recognizable as extrapolations of them.

The system is likely to include a long-term memory component, probably represented as some form of graph structure and/or as a set of production rules. If the long-term memory is modeled as a graph structure, nodes within it may contain relatively small amounts of typed data or (hierarchical) structures of lower level nodes. The system should include a working memory, either as a separate component or as that (small) part of long-term memory that is currently activated. And it should include either a separate processor component or, more likely, a set of processes that operate on the activated contents of working memory.

Higher level tasks — for example, problem-solving — are likely to be performed within the context of one or more problem spaces or some similar architectural construct that includes a goal, a scheme or data type in which to represent problems, and a data structure consistent with that representation. To chart an overall approach to a task and to respond to problems or altered conditions within particular contexts, the system should include sophisticated strategies and tactics. Although solving a problem the first time may involve considerable trial and error, with experience, the system should learn to recognize problems and to retrieve solutions worked out earlier for similar problems.

Finally, a complete cognitive system would also include receptor and effector functions that interact with the outside environment, although these functions have largely been ignored in IPS models and architectures.

Specialized IPS Models

In the preceding discussion, I reviewed three general cognitive models and architectures. These systems have been most successful at modeling or simulating isolated forms of human behavior, such as memory access or stimulus-response behaviors; at solving well-defined problems, such as cryptarithmetic and chess; and at solving problems in which weak principles of strategy, such as hill climbing and means–ends analysis, are sufficient. For problems of this type, one can define a set of rules that can be used in a systematic way to carry out the task, and one can tell unambiguously whether or not a solution has been reached. Consequently, these models have routinely been expressed as computer simulation programs.

They have been less successful at modeling or simulating coherent behaviors that extend over long periods of time; at solving poorly defined problems that require judgment and/or qualitative evaluation; and at performing tasks that require complex strategies or interactions with other individuals. Knowledge-construction tasks — such as writing documents and computer programs, designing a building, or planning a sales campaign — are typical of this type of activity. The rules for performing these tasks are usually general — more like rules of thumb rather than precise procedures that can be systematically applied — and no rule exists to tell exactly when the task is complete. They also require strategies that extend over hours, days, or even longer periods of time. Currently, the state of the art does not support models for extensive conceptual construction tasks that have the rigor and consistency of those for well-defined problems. Imagine, if you will, what would be required to develop a simulation system that could write a journal article, a proposal, or even a letter to a friend.

Nevertheless, significant progress has been made over the past decade in understanding complex, real-world forms of cognition. Researchers have developed specialized models and frameworks that apply to specific tasks, such as expository writing, or to specific situations, such as carrying out a particular task using a particular computer system. They have incorporated concepts from the more general systems and architectures discussed previously, demonstrating the applicability of those concepts to ill-defined problems. But they

have also been forced to extend those concepts, change their points of view, or accept limitations in their research agendas.

In this section, I extend the discussion of basic components for cognitive systems by looking at the extensions and adaptations made in three specialized models or frameworks. The first is Dick Hayes' and Linda Flower's cognitive model for expository writing. The second is the Card, Moran, and Newell models of human–computer interaction. The third is an architectural framework developed by our group that combines aspects of both the Hayes and Flower and the Card, Moran, and Newell models. In all three discussions, I emphasize the steps taken by these researchers that have enabled them to address these more complex tasks and situations.

Writing

As noted previously, I consider writing to be the quintessential conceptual construction task. First, writing requires a number of different intellectual skills used to transform an inchoate, loosely structured network of concepts into a well-structured, clearly expressed document. Thus, it is an inherently complex process that is made more so by extrinsic and document-specific factors — such as interruptions and the availability of necessary information. Second, the vast majority of tasks that involve building complex structures of ideas express those ideas as some form of document. Thus, writing is an integral part of most knowledge-construction tasks. Third, at a sufficiently abstract level, many of the processes and strategies used to plan, write, and revise documents can also be used to plan, express, and refine other types of information. This is equally true for collaborative as well as individual work. Thus, if we can understand the cognitive and social processes involved in writing, we will be well on our way to understanding a number of other tasks, as well.

During the past 15 years, a great deal of research has been done concerning the mental behavior of writers. The most important and most influential body of work is that of Dick Hayes and Linda Flower. In this section, I discuss their cognitive model of writing. In doing so, I want to emphasize two points: the IPS basis of their model, and, second, the adaptations and changes in research perspective they made that enabled them to address a task as ill-defined as writing.

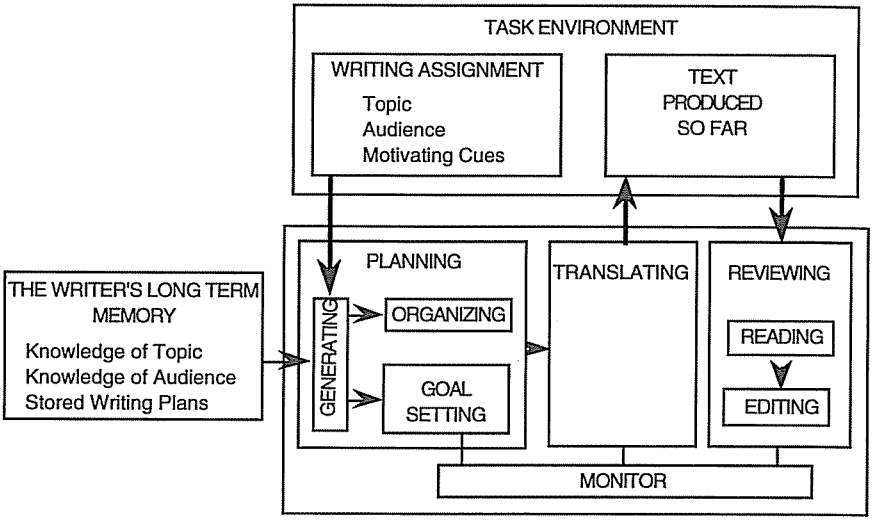

Fig. 4.4. Hayes and Flower's cognitive process model of writing. (Adapted by permission from Hayes & Flower, 1980.)

Hayes and Flower studied expository writing, as done by both students and adult professionals. Because they are colleagues at Carnegie Mellon University, it is not surprising that many of their methods and concepts were based on the earlier IPS model and the methodology developed by Newell and Simon to study human problem-solving behavior (Hayes & Flower, 1980; Flower & Hayes, 1984). For example, the Hayes and Flower writing model adopts a basic information processing perspective, and they asked writers to think aloud as they planned, wrote, and revised texts, just as Newell and Simon asked subjects to think aloud as they played chess or solved cryptarithmetic problems.

The Hayes and Flower model of writing is shown in Fig. 4.4. It includes a representation of the task environment that includes "the problem" to be solved — in this case, the writing assignment. It contains a long-term memory component that includes several types of semantic knowledge relevant to the task. The third major component of the model, shown at the lower right of the figure, is not labeled; presumably, it is a processor component because it contains three high-level processes and their associated subprocesses. Planning

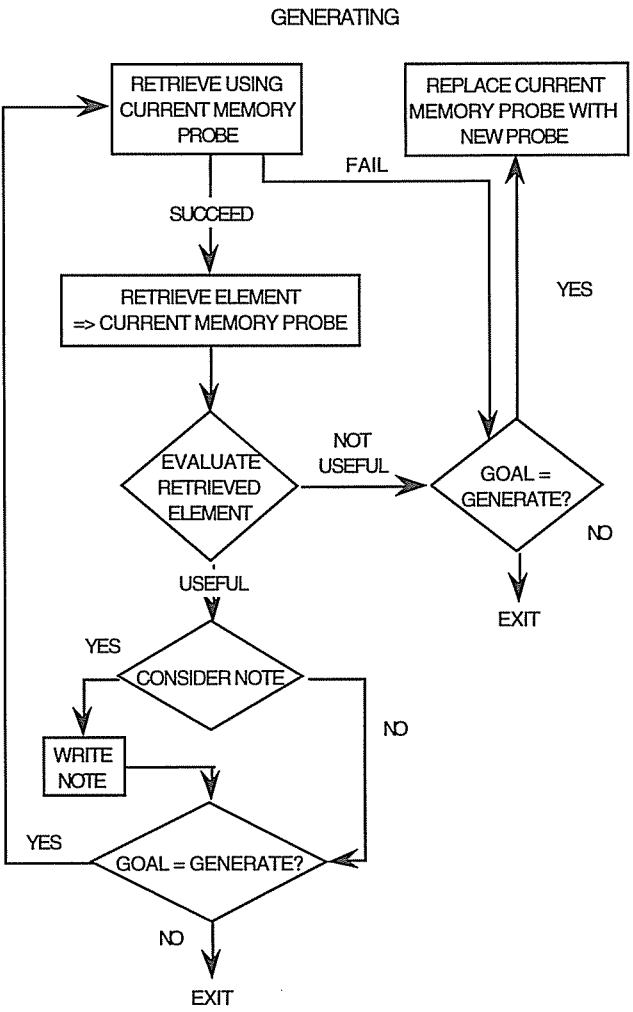

Fig. 4.5. Hayes and Flower's Generating (sub)process. (Adapted by permission from Hayes & Flower, 1980.)

includes three subprocesses: memory access (generating), a process for constructing a plan for the document expressed in some schematic form (organizing), and a goal-setting component to control movement among these processes. Translating, which contains no subprocesses, is concerned with encoding the ideas produced during planning into continuous prose — the text produced so far. Finally, reviewing

involves two subprocesses: reading the text produced so far and editing it. The monitor is a different type of process, which I will discuss in just a moment.

Processes are represented in the model as decision procedures, expressed as flow charts. Fig. 4.5 shows the flow chart for generating, a subprocess that is part of the larger planning process. It is driven by three operations. First, long-term memory is accessed using the current contents of working memory as a cue. Second, an evaluation is made to determine whether or not the retrieved concept is useful. Third, if the concept is useful, it is externally represented as some form of note — Hayes and Flower's term for any type of information other than sustained prose, such as a word, phrase, symbol, and so forth. This (sub)process continues so long as goal = generate. When the goal changes, a new process is engaged.

Overall control of the writing system is provided by the monitor. It engages the different processes in accord with a set of strategies using a goal/subgoal mechanism, similar to that found in general IPS models and architectures. It differs from those systems, however, in storing only the current goal, rather than a hierarchy of goals/subgoals. The control program that runs in the Monitor is shown in Fig. 4.6. It is defined as a sequence of production rules. All decisions are based on the current contents of working memory. When working memory contains continuous prose, the monitor engages the edit process (rule 1). When it contains abstract information, the generate process is engaged (rule 2). When it simply contains a goal, the corresponding process used to achieve that goal is engaged (rules 7–10). Goal-setting productions that change the current goal (acted on in rules 7–10) appear as rules 3–6 in the program. Here, individual differences among writers come into play. Hayes and Flower describe four different strategies found in the writers they studied. Each strategy, which they refer to in the figure as a configuration, consists of an alternative sequence of rules 3–6. These alternative strategies are inserted into the overall control sequence in accord with the individual writer's personal writing habits.

Hayes and Flower used their model as a framework with which to analyze specific writing behaviors. They typically asked writers to think aloud as they plan and write their documents. Human judges then code the individual statements in the transcribed protocol to identify which cognitive (or metacognitive) process is currently

```
1.    [Generated language in STM → edit]
2.    [New information in STM  → generate]
3.-6. Goal setting productions (These vary from writer to
      writer; see different configurations, below.)
7.    [(goal=generate) → generate]
8.    [(goal=organize) → organize]
9.    [(goal=translate)→ translate]
10.   [(goal=review)  → review]

Configuration 1 (Depth first)
3.    [New element from translate      → (goal=review)]
4.    [New element from organize       → (goal=translate)]
5.    [New element from generate       → (goal=organize)]
6.    [Not enough material             → (goal=generate)]

Configuration 2 (Get it down as you think of it, then review)
3.    [New element from generate       → (goal=organize)]
4.    [New element from organize       → (goal=translate)]
5.    [Not enough material             → (goal=generate)]
6.    [Enough material                 → (goal=review)]

Configuration 3 (Perfect first draft)
3.    [Not enough material             → (goal=generate)]
4.    [Enough material, plan not complete→ (goal=organize)]
5.    [New element from translate      → (goal=review)]
6.    [Plan complete                   → (goal=translate)]

Configuration 4 (Breadth first)
3.    [Not enough material             → (goal=generate)]
4.    [Enough material, plan not complete→ (goal=organize)]
5.    [Plan complete                   → (goal=translate)]
6.    [Translation complete            → (goal=review)]
```

Fig. 4.6. Hayes and Flower's monitor control program, with four different strategies used by individual writers. (Adapted by permission from Hayes & Flower, 1980.)

operating. Figure 4.7 shows a single writing session for a subject analyzed in this way. Each think aloud statement is shown at the time it occurred and classified as one of four processes. The session can be divided into three large sections in which planning, writing, and revising processes dominate, respectively. Using analytic diagrams such as this, Hayes and Flower were able to manually trace the writing behaviors of their subjects through the various levels of their model, confirming that the model could account for the strategies and shifts from process to process exhibited by them.

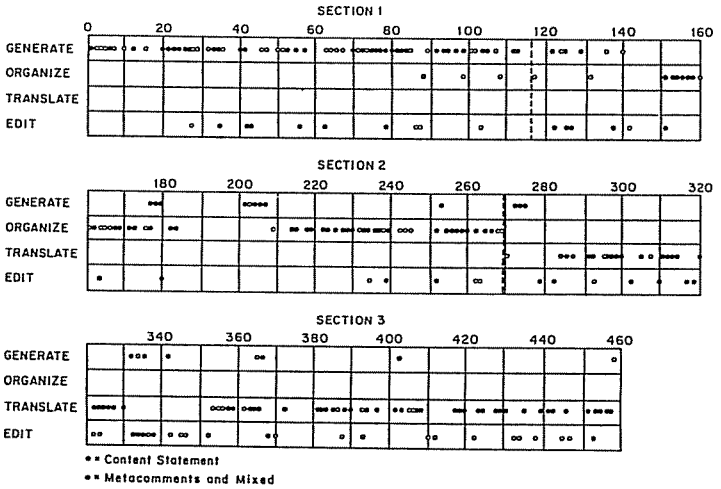

Fig. 4.7. Sequence of processes generated by one subject during a writing session. (Adapted by permission from Hayes & Flower, 1980.)

Such analyses led to a number of insights into the writing process that were not incorporated into the model, per se. For example, in a study of the differences in strategies between expert and novice writers, they found that the goals structures generated by the experts were so much more extensive and complex than those of the novices, the two groups of writers could be viewed as carrying out different tasks, rather than exhibiting different strategies for the same task (Hayes & Flower, 1986).

In summary, the Hayes and Flower model has much of the look and feel of the formal IPS models and architectures discussed in the preceding section. It consists of three major types of components: the overall framework for the model, represented as a box structure; specifications for individual processes, represented as flow charts; and a control procedure, represented as production rules, that includes alternative strategies. It describes both the overall writing process common to all writers as well as specific strategies used by individual writers, and a human interpreter can trace the behavior of an individual writer through the model. Research that has used the model as a framework for analysis has led to a number of interesting and

useful insights that have enriched the model without being incorporated into it.

Although the Hayes and Flower model offered a much more detailed view of the writing process from what existed prior to its publication, one must keep in mind its limitations. The model is not sufficiently precise that it could be implemented as a simulation program or that it could be used to generate strong predictions of writers' behaviors. This is not a deficiency but, rather, indicates a different set of assumptions and goals. It was intended to be an analytic and conceptual model, to guide studies of individual writing behaviors, and to help researchers understand this particularly complex intellectual process. It took this approach to research because it promised more interesting and more practical results, given current knowledge of the task. And it has delivered good results. At some future time, using what we learn from this and other similar analytic models, we may eventually be able to build interesting predictive models or simulation systems for writing, but that time seems distant.

Finally, granting Hayes and Flower the informality of their approach, basic components of the model need to be defined more precisely. For example, process is not defined explicitly but, rather, by example through descriptions of specific processes. If we step back and ponder their underlying form, processes appear to be large-grain structures similar to problem spaces, as opposed to the small-grain cognitive operations or methods that occur within those spaces in other IPS systems. Each process is associated with a goal that causes the monitor to invoke that process. Processes include subprocesses. And they normally produce some form of intellectual product, such as a note that represents a concept retrieved from long-term memory, or a change in an existing product, such as an editorial change made to the Text Produced So Far. However, both the data types and structures used within processes are left as informal concepts, such as the traditional notion of a text, rather than as formally defined components of the model. Thus, although processes seem very similar to problem spaces, the relationship between the two is not spelled out. What is needed is an additional level of detail that makes explicit the basic architecture on which the model is built.

Human–Computer Interaction

A second application of IPS concepts to a specialized task or situation is models of human–computer interaction. Just as writing is a part of many collaboration tasks, so an increasing number of groups use computer systems to help them with their work. Consequently, this work is also highly relevant for a concept of collective intelligence. In this section, I describe a particular view of human–computer interaction (HCI) research that has dominated the field for nearly a decade.

Card, Moran, and Newell (1983) defined a general framework in which specific models can be developed that characterize the behaviors of users performing specific tasks with specific computer systems. The framework has two major parts. Their model human processor (MHP) provides both a cognitive architecture and a set of quantitative measures derived from the literature for a range of basic processes and motor actions required to work with a computer. The second part of the framework is a set of categories for describing computer-related tasks. Because Allen Newell was one of the co-authors, it is not surprising that this framework resembles both the earlier Newell and Simon IPS model and Newell's more recent Soar architecture.

The model human processor, shown in Fig. 4.8, includes the following architectural components: a long-term memory, a working memory that includes separate stores for visual and auditory information, perceptual and motor processors, and a cognitive processor. A key aspect of this model is the set of parameters derived from the literature that quantize both the capacity of components (e.g., the amount of information that can be held in working memory) and the time required for fine-grained cognitive actions (e.g., the amount of time required to access long-term memory or to move the eyes). Thus, if one can decompose a task into a sequence of basic cognitive and motor actions, then one can use the model to predict the time that will be required for someone to perform that task.

The basic MHP model is extended to include some 10 higher level regularities, called principles of operations. Two examples are the *power law of practice*, which describes the time expected for a user to perform a given task after a varying number of practice trials, and *Fitt's law*, which describes the relationship between the time required

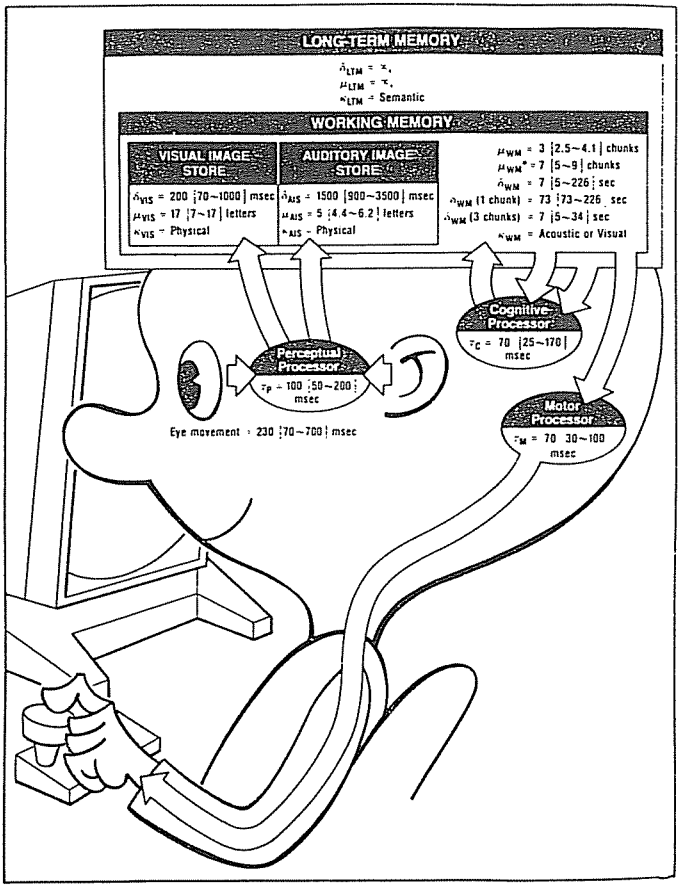

Fig. 4.8. Card, Moran, and Newell's Model Human Processor. (Reproduced by permission from Card, Moran, & Newell, 1983.)

by a user to move his or her hand a given distance toward a target and the size of that target. These regularities provide larger behavioral structures in which to fit the more basic processes and components of the MHP architecture.

The second major part of the Card, Moran, and Newell framework is a set of categories, called GOMS, that is used to define models for tasks performed with a given computer system. GOMS is

an acronym for *goals, operators, methods,* and *selection rules*. A task typical of those that have been described in terms of GOMS categories is a user of a particular computer editor making changes to a document from a previously marked copy of the text (Card, Moran, & Newell, 1983). Tasks are described in terms of a set of goals. Higher level goals generate lower level goals; for example, a goal of making a given change in the computer version of the text as indicated in the marked paper version might generate the subgoal to find the appropriate location in the computer version where the change is to be made. Thus, tasks produce hierarchies of goals and subgoals, similar to those produced in the other IPS models and architectures discussed above. Operators are basic system actions, such as a menu selection or a single typed command. They represent the user's knowledge of how the system works. Methods are short sequences of operators known to the user that are routinely used to accomplish a goal. Thus, for example, inserting a word in a line may require moving the cursor, typing a control sequence, typing the character string to be inserted, followed by typing another control sequence to conclude the task. Sequences of operations such as these may be used so often that they become automatic.

The most complex part of the GOMS model is the set of selection rules. Normally written as production rules, they identify a set of conditions and the action that will follow when those conditions are met. Thus, they predict users' behaviors under different conditions. Because goals may be part of both the conditional and action parts of a selection rule, they can generate hierarchies of goals and subgoals that eventually terminate in specific methods defined as sequences of basic operations. Consequently, by applying the timing parameters from the model human processor model, one can also predict the amount of time these sequences will take, and hence the task as a whole.

The selection rule component has been extremely influential in determining the kinds of problems GOMS can and cannot address. GOMS has been used most successfully for predicting short, independent sequences of actions, ranging from three or four to a dozen or so. Tasks comprised of sequences of this length are usually measured in seconds or tens of seconds. Typical examples, in addition to the editing task described previously, include performing independent operations with a spreadsheet program (Olson & Nilsen, 1988) and playing video games (John & Vera, 1992).

Two recent extensions to the basic GOMS model have extended this limit. The first, called CPM-GOMS, includes a critical path

component and has been used to model the behavior of long distance telephone operators (Gray, John, & Atwood, 1992). It enabled researchers to predict behaviors of operators using two different computer workstations for transactions involving 15 to 20 operations and lasting about as many seconds. A second extension, called Browser-Soar, combines the basic GOMS approach with the problem space concept from Soar. Developed by Virginia Peck and Bonnie John at CMU, Browser-Soar has been used to model independent browsing operations performed by users of a help system lasting for approximately 100 seconds and involving several subtasks (Peck & John, 1992).

Although CPM-GOMS and Browser-Soar significantly extend the GOMS approach, the tasks they model are quite different from the sustained, interdependent activities that occur in knowledge-construction tasks, such as writing. Consider what would be involved in developing a detailed GOMS model for expository writing, comparable to the Hayes and Flower model. This would require predicting the order in which a writer will engage the different processes identified in the model: when and under which conditions he or she will switch from one process to another; when a particular line of thought will run dry; when someone will hear a stray conversation in the background and be reminded of an idea that can be incorporated into the document; and so on. A predictive model that could handle behaviors of this sort would constitute an artificial intelligence capable of simulating expository writing. This is not to say that all aspects of complex behavior are unpredictable. Indeed, we can predict behaviors such as the average length of time spent in continuous work episodes based on attention span and limits of working memory. But predicting specific cognitive behaviors that are affected by complex strategies and tactics, the semantics of the task domain, and extrinsic factors lies far beyond current capabilities.

This difference between a GOMS approach and a Hayes and Flower approach is a paradigmatic difference. The first is quantitative and formal and assumes that the ultimate value of a model, as well as its verification and refinement, are based on its capability to generate predictions. The second is qualitative and informal and assumes that the value of a model lies in its capability to provide interesting and useful insights into complex, real-world tasks. The second does not deny the value of the first; it just defers it until such future time when the base of knowledge is sufficient that generative models can be developed that can handle the same factors and situations. Although

the Hayes and Flower model was not entirely satisfactory, it illustrates the kind of model that is likely to be most practical and most useful for current considerations of collaborative groups.

Although the MHP/GOMS framework is based on IPS concepts, it is not as developed. Like other IPS systems, it includes goals, goal/subgoal hierarchies, and a set of basic operations. These components are situated within the user's long-term and working memories. But the framework does not include the additional structure provided by a problem space. Thus, it includes no inherent concept of data type or data structure. Such concepts may be part of the computer system presumed by a GOMS model, but direct references to conceptual data must be incorporated into other components of the framework. Frequently, goals have served this purpose. But this overloads the concept: Sometimes goals refer to abstract intentions, at other times to changes in the system's data model that will realize those intentions. Given Soar's subsequent inclusion of both a problem space and its associated functions as basic components of the human cognitive architecture, omitting some form of larger context in which cognition is presumed to take place appears to be a severe limitation in GOMS.

Finally, the MHP/GOMS approach has been used most frequently to characterize the behavior of users working with existing computer systems. It has not been used extensively as a tool for designing new or improved systems. One reason for this is that there is no independent cognitive component in the framework. As a result, task descriptions are normally cast in terms of the options offered by an existing computer system — operators or sequences of operators — for accomplishing a basic task goal. Consequently, there is no natural mechanism in which to develop a separate cognitive model for a task that can then be mapped onto a separate system model that, in turn, can inform subsequent design and implementation of that system. If we wish to develop computer systems that closely match users' cognitive behaviors and strategies, as is the goal for *intelligence amplification* systems, we will need capabilities that go beyond those included in the MHP/GOMS framework.

Cognitive Modes and Strategies

In this section, I discuss a framework developed by our research group that can be used both to describe complex, real-world

knowledge-construction tasks and to develop theory-based computer systems to support those tasks. Its basic constituents are a set of *cognitive modes* used by individuals to perform a given task and the *strategies* they use to guide them as they shift from one mode to another in carrying out that task.

The approach combines aspects of both the Hayes and Flower and the Card, Moran, and Newell models/frameworks. Like the Hayes and Flower approach, the mode/strategy approach is oriented toward analysis and description, rather than prediction; thus, it, also, bypasses the limiting factor in the GOMS approach — selection rules — by regarding users' strategies as the object of discovery rather than a constituent that must be in place prior to analysis. Consequently, mode-based models can address coherent, interdependent behaviors that extend over hours, days, and even longer periods of time. Like the Card, Moran, and Newell approach, the mode/strategy framework is a well-defined architectural construct; thus, it avoids the ambiguity of the Hayes and Flower process model. However, unlike MHP/GOMS models, mode/strategy models can be mapped directly onto system design. Thus, the mode/strategy framework retains key features of these other approaches while addressing several of their limitations.

In this section, I first define the mode/strategy framework as a general architectural construct and then illustrate its use by describing a set of modes for expository writing. After that, I describe a writing support system whose design was based upon that set of modes. Finally, I describe a mode-based analytic model developed by our group for studying the cognitive strategies of writers using the system.

Mode/Strategy Framework

A *cognitive mode* is a particular way of thinking used for a particular purpose. For complex tasks, such as conceptual construction tasks, human beings engage different cognitive modes in order to accomplish different parts of the task. Consequently, they move from one mode to another in accord with strategies they know for the task and in response to changing conditions in both the content domain and the external environment.

A mode is determined by four factors: goals, products, processes, and constraints. Thus, a particular mode of thought is associated with a particular goal that will be realized by producing a particular type of

conceptual product, drawing on particular cognitive processes in accord with a particular set of constraints (Smith & Lansman, 1989). By implication, a given mode may exclude or discourage certain kinds of products and the processes used to develop those products. Let's look more closely at each of these four factors that identify a given mode. Although the discussion is general, I use expository writing as the example task. Other tasks would include other sets of modes, but those modes would be cast within the general framework described here.

Goals represent the intentions that lie behind a person's use of a given mode. Thus, for example, one may engage a brainstorming mode as part of the overall task of writing in order to gain a general sense of the information that is available to the writer. A goal is normally realized by creating some form of conceptual product or by making a change in one or more existing products, but the two are not the same: Goals are abstract; products are concrete.

Different cognitive modes provide different options for representing concepts or structures of concepts. Consequently, different modes support the development of different types of information *products*. These include words, phrases, sentences, outlines, diagrams and drawings, symbols, equations, computer code, and other forms. Thus, product encompasses both the data type and the data structure components of the problem space architecture.

Cognitive *processes* act on cognitive products to create them, to extend or modify them, or to transform one type into another. For example, some processes are concerned with accessing and representing a portion of the associative or semantic network of an individual's long-term memory. Others are concerned with transforming that network into a hierarchy. Still others are concerned with translating abstract ideas into specific representations, such as sentences or diagrams. Consequently, certain processes are favored in particular modes, whereas other processes are de-emphasized or even suppressed. Thus, process is a small-grain concept in the modes framework, comparable to operator in GOMS and IPS systems, but distinctly smaller than the process found in the Hayes and Flower model.

Constraints determine the choices available within a mode. They set thresholds on evaluation functions that guide the flow of conceptual thought. Examples of these functions, which usually operate automatically, include determining the relevance of a retrieved

concept, evaluating a change made to a conceptual structure, and invoking or suppressing corrective processes. Constraints are raised or lowered in different modes in accord with the general goal or purpose for engaging that particular way of thinking. Thus, they set the overall "tone" for a mode. None of the systems discussed previously include constraints as a basic component of the architecture, although a limited form of constraint is implicit in the evaluative component of basic strategies, such as means–ends analysis, used in IPS problem spaces.

To gain a better feel for the way interdependent combinations of these four factors determine particular modes of thought, let's look more closely at two specific modes used by many writers: *exploration* and *organizing*. During exploration, the goal is to externalize ideas and to examine those ideas in different combinations and relationships with one another. Consequently, constraints are kept to a minimum to encourage creativity and multiple perspectives. For example, during this mode, many writers pay little attention to spelling, neatness, or syntactic precision; they may even suppress their tendencies to notice these things. They may also try not to make decisions about which ideas are relevant or not relevant in order to generate a larger pool of possibilities from which to eventually select. As a result, the products generated are often informal, consisting of notes, jottings, diagrams, loose networks of concepts, and so on. Processes include recalling concepts from memory, basic encoding, associating and relating concepts, and building small component structures.

During organizing, the goal is to plan the actual document to be written. Consequently, constraints are tightened and thinking becomes much more rigorous and systematic. The goal is usually achieved by creating some concrete representation of a plan for the document, such as an outline, tree, or other form of hierarchical structure. The processes emphasized in this mode are those needed to construct the plan. They include analyzing; synthesizing; noting various relationships between ideas, such as cause and effect or subordinate–superordinate relationships; and comparing different parts of the evolving plan for consistency and parallel structure.

Thus, exploration and organizing are distinctly different ways of thinking. And they differ from other activities such as translating the abstract ideas in the plan into sentences or editing the resulting document. Figure 4.9 provides a more complete set of modes for expository writing. In addition to exploration and organizing, it

	Processes	Products	Goals	Constraints
Exploration	•Recalling •Representing •Clustering •Associating •Noting subordinate-superordinate relations	•Individual concepts •Clusters of concepts •Networks of related concepts	•Externalize ideas •Cluster related ideas •Gain general sense of available concepts •Consider various possible relations	•Flexible •Informal •Free expression
Situational Analysis	•Analyzing objectives •Selecting •Prioritizing •Analyzing audiences	•High-level summary statement •Prioritized list of readers (types) •List of (major) actions desired	•Clarify rhetorical intentions •Identify and rank potential readers •Identify major actions •Consolidate realization •Set high-level strategy for document	•Flexible •Extrinsic perspective
Organizing	•Analyzing •Synthesizing •Building abstract structure •Refining structure	•Hierarchy of concepts •Crafted labels	•Transform network of concepts into coherent hierarchy	•Rigorous •Consistent •Hierarchical •Not sustained prose
Writing	•Linguistic encoding	•Coherent prose	•Transform abstract representation of concepts and relations into prose	•Sustained expression •Not (necessarily) refined
Editing: Global Organization	•Noting large-scale relations •Correcting inconsistencies •Manipulating large structural components	•Refined text structure •Consistent structural cues	•Verify and revise large-scale organizational components	•Focus on large-scale features and components
Editing: Coherence Relations	•Noting coherence relations between sentences and paragraphs •Restructuring to make relations coherent	•Refined paragraphs and sentences •Coherent logical relations between sentences and paragraphs	•Verify and revise coherence relations within intermediate sized components	•Focus on structural relations among sentences and paragraphs •Rigorous logical and structural thinking
Editing: Expression	•Reading •Linguistic analysis, transformation, & encoding	•Refined prose	•Verify and revise text of document	•Focus on expression •Close attention to linguistic detail

Fig. 4.9. Seven cognitive modes for expository writing, including the processes, products, goals, and constraints for each mode.

includes *situational analysis*, in which the rhetorical context of the document is explored; *writing*, per se; and three modes for *editing*: *organizational* editing, involving large structural components of the document; *coherence* editing, involving relationships within individual paragraphs or small sections of the document; and *expression* editing, involving individual sentences (or other basic types of information).

The preceding discussion described cognitive modes as separate "islands" of thought to emphasize their distinctness. However, modes are also related to one another in fundamental ways. First, individuals shift from one mode to another over time; consequently, they exhibit different tendencies or patterns of behavior in the order in which they engage different modes and the conditions that cause them to shift from one mode to another. Second, the intellectual products created in one mode are often carried to or appear in another mode in which a different set of processes is used to continue development or to transform one type of product into another.

The first relationship is concerned with the *strategies* and *tactics* an individual uses to accomplish a task. Strategy refers to an individual's overall understanding or image of a task and the large-grained process that person has learned or developed that enables him or her to accomplish that task. Examples of strategy include the "stages" model of writing and the "waterfall" model of software development. Tactics refer to the shifts people make from one mode to another in order to respond to problems that arise or to changes in conditions. For example, writers may return to organizing mode when they realize during writing that the plans they constructed earlier have problems (Hayes & Flower, 1980). Thus, modes help individuals focus their attention on a single activity at a time, whereas strategies and tactics provide them with the means to move from one activity to another in coherent ways. Of course, not everyone uses the same strategy for a given task; in fact, differences in individual behavior can be characterized in terms of patterns in the sequences of modes they engage (Lansman & Smith, 1993).

The second relationship is concerned with the transfer of information from one mode to another. When an individual follows a global strategy for a task, he or she normally produce a *flow* of intermediate products in which the output of one mode becomes the input for another. For example, during exploration, many writers represent concepts externally, cluster them, and then link them into a loose network of associations. During organizing, they transform this loose network into a coherent, consistent structure for the document.

If they use an outline or tree for this purpose, then the transformation is from a network to a hierarchy. During writing, they transform abstract concepts and relations in the plan into continuous prose, graphic images, or other types of information. During editing, they refine the structure and expression of a draft document to produce an improved or final version.

This flow of products, however, is not one-way and continuous. As an individual shifts from one mode to another, intermediate products flow back and forth, as well. For example, writers may find while organizing that they do not have crucial information needed for a particular section. Rather than interrupt their thinking to seek out that information then, they may decide to continue organizing but leave the relevant section undeveloped. Later, when the missing information is available, they may interrupt their writing, revert to organizing and/or exploration modes to build the missing portion of the document's structure. When the missing part has been filled in, they resume writing (Smith & Lansman, 1991).

In summary, for many intellectual activities, individuals divide tasks into subtasks, set goals and subgoals, produce intermediate as well as target products, and employ different processes to produce them. They use general strategies to guide overall behavior and more specific tactics to resolve problems. The behavior of particular individuals or groups carrying out specific tasks can be modeled by identifying the particular set of cognitive modes they use along with their strategies and tactics. Thus, the general concepts of *mode, strategies,* and *tactics* can be viewed as an architectural framework that can be applied to a broad range of tasks.

Mode-Based Writing Environment

If a task is described in terms of a set of cognitive modes and the expected flow of intermediate products from one mode to another, that description can be used in a direct and natural way to guide design of a computer system to support that task. This is done by including different working contexts for the different cognitive modes and providing mechanisms for moving data from one context to another. (Smith & Lansman, 1992).

More specifically, after identifying a set of cognitive modes for the task, one can then design a corresponding set of interface or *system modes*. Each system mode can be associated with a different window in the user interface. Each supports a different data model that includes both data types and data structures appropriate for its corresponding cognitive mode(s). Each system mode should also include a set of functions sufficient to construct data objects of that type. Ideally, these functions would be presented in one-to-one relationship with corresponding processes in the cognitive mode; however, in some cases, several system operations may be required to represent the results of a single cognitive process. Thus, in general, there is a many-one relationship between cognitive process and system actions.

To support the flow of intermediate products from one cognitive mode to another, the system should provide functions for copying or moving data from one system mode to another. An alternative design would be to overlay the data with a succession of modes, rather than to move data between modes; however, this approach is neither as general nor as flexible as the flow model and can lead to problems, such as blocking shifts between pairs of modes that have distinctly different data types/structures.

An example system whose design was based on a mode/strategy task model is the Writing Environment (WE). Built by our research group, WE includes four system modes that support six of the seven cognitive modes included in Fig. 4.9 (Smith et al., 1987). A sample screen for WE is shown in Fig. 4.10.

Network mode, shown in the upper left window, supports exploration. The underlying data model is a directed graph embedded in a two-dimensional space. Thus, the user has maximum flexibility for representing concepts as nodes (boxes with a word or phrase to express the idea), moving them to form clusters of loosely related ideas, and linking them to denote more specific relationships. Small conceptual structures can also be built here and later used in other modes. Although writers may do all of their large-grain structural work in this mode, the system provides another mode that is better suited for building the actual plan for the document.

Tree mode, shown in the lower left window, supports the organizing cognitive mode. The underlying data model is a tree or hierarchy, and the functions provided for building the tree are

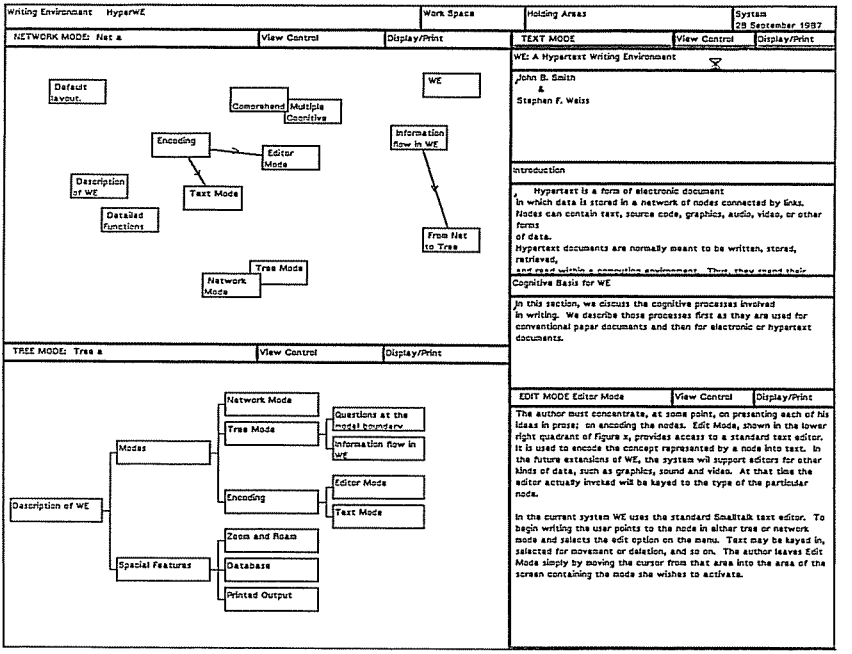

Fig. 4.10. Sample screen for the Writing Environment (WE), showing network, tree, editor, and text system modes.

constrained so that one cannot denote a relationship that would violate the integrity of the hierarchical structure. Thus, the functions and constraints of tree mode are different from those of network mode in which maximum flexibility is emphasized. This design decision was based on research in reading comprehension that shows that, in general, hierarchical documents are more easily and more accurately comprehended than unstructured documents or documents with other structures (Smith & Lansman, 1989). Although users may create the tree from scratch, more often they combine creating new nodes in tree mode with copying nodes or small component structures from network mode. Thus, system design encourages writers to transform the loosely structured network of ideas, developed in

exploration/network modes, into consistent, well-defined hierarchical structures in organization/tree modes. But it does not require them to do so.

The writing cognitive mode is supported by *editor* mode, shown in the lower right corner. At any time in the overall writing process, users can open a node in either the network or tree modes and write a block of text in the editor that will be associated with that node. Because writers work on the contents of a single node at a time, the design of the system encourages them to focus their attention on a single concept and to transform that abstraction into linguistic expression. Thus, at any given time, the writer is concerned with writing a small, manageable component as opposed to writing the entire document, thereby simplifying the overall writing/translating task. To construct a conventional linear document from all these pieces, the system "walks" the tree — top to bottom, left to right — and gathers up the contents of the various nodes. It then sends these concatenated pieces to a file or printer.

Finally, the upper right system mode, called *text* mode, is intended for coherence editing. Although the system reduces users' cognitive load by enabling them to work with the structure of the document in abstract, schematic form (tree or network) and to divide the actual writing of the document into a succession of small writing subtasks, this approach can lead to documents with inconsistencies and awkward transitions between sections. To address these problems, text mode presents the document in its linear, concatenated form for editing. Thus, one can read the text continuously "across" node boundaries, check transitions, move sentences from one node to another, and so forth.

Thus, WE's four system modes — network, tree, editor, and text — correspond to four cognitive modes — exploration, organizing, writing, and coherence editing. For organizational editing, writers use tree mode — by moving branches and nodes around in the tree, they can reorganize the text of the associated document. To support expression editing, writers use either editor or text modes. Consequently, six of the seven cognitive modes described in Fig. 4.9 are supported by the four WE system modes. WE does not support situational analysis mode in which writers analyze the rhetorical context and make strategic decisions about their documents; however, we have developed heuristics to assist with this process (Smith & Smith, 1987), which could be incorporated into future versions of the system.

Ideally, development of the task model precedes system design. In practice, development of the two may be an iterative process, with work in down-stream phases — for example, system building — informing work in up-stream phases — for example, model building. Ultimately, it is not important which came first, but that the two end up being consistent with one another. The mode/strategy framework can contribute to this process by making the relationships between the two straightforward.

Mode-Based Analytic Model

In the preceding sections, I showed, first, that knowledge-construction tasks, such as expository writing, can be described in terms of a set of cognitive modes, the transitions that occur between modes, and the resulting flow of intermediate products from one mode to another; and, second, that mode-based task descriptions can be mapped onto system design in a natural and straight-forward way. In this section, I take this synthesis one step further by introducing the notion of an analytic model that takes into account the mediating effects of such a system on task behavior. Here, I only introduce the idea in order to complete this picture of the mode/strategy approach; in chapter 7, I describe several such models in more detail in discussing the general concept of *strategy*.

In order to study the cognitive behaviors of individuals using the WE system to plan and write documents, we instrumented the system so that each time a user selected a data object, chose a menu option, or moved from one system mode to another, a record of that action was recorded. The sequence of all such records for a session comprise what we call an *action level protocol.* These data provide a detailed record of the system functions used to represent the results of users' cognitive processes. They also identify cognitive products produced by those processes and their evolution over the course of the task. Thus, they provide a record of the material production of a document, from earliest brainstorming and planning to final editing. However, because not all thought results in a system action, we cannot claim that the record is complete or that it includes no distortions, but because system model and task model are closely related, the action protocol should represent to an approximation a trace of the user's thought process during the task.

Analytic models can be developed to analyze these protocol data and to uncover patterns in users' cognitive behavior. We chose to express these models as grammars. A conventional grammar takes as input a string and determines whether or not that string is contained in a language. Thus, for a natural language such as English, a grammar takes as input a string of words and determines whether they constitute a valid sentence in English. To do so, the grammar produces a parse of the sentence that shows its grammatical structure. An analytic model that is expressed as a grammar takes as input an action level protocol and produces as output a parse tree that describes the user's strategy for the session.

We developed several models that can analyze writers' strategies. One produces parse trees that include six hierarchical levels (Smith, Rooks, & Ferguson, 1989). The root of the tree is the *session*. Each session is divided into a sequence of cognitive *modes*. Each mode, in turn, is divided into a sequence of cognitive *processes*. Each process symbol is then linked to one or more cognitive *products*. Thus, the top four levels of the model are cognitive and, hence, independent of the WE system.

The bottom two levels of the grammar map the cognitive portion of the model onto the design of a specific computer system — in this case, WE. To represent a change to a cognitive product, users perform one or more system *operations*, each of which requires several system *actions*. Because each protocol record corresponds to a single system action, actions are the terminal symbols in the grammar, analogous to individual words.

This grammar is expressed as a set of production rules, supplemented by some half-dozen functions that recognize particular context-sensitive relationships. It is implemented as a computer program; thus, it can be applied consistently and automatically across multiple user protocols. We used the parser to analyze a number of user sessions under different experimental conditions. Results of these studies are described in Smith and Lansman (1989), Lansman (1991), Smith and Lansman (1992) and Lansman and Smith, (1993).

Perspective

Like the IPS architectures discussed previously, the mode/strategy framework provides an architectural construct that applies to a wide

variety of tasks. It includes an explicit context in which cognition takes place as well as a set of components that occur or operate in that context. It differs from them, however, by including constraints and a richer set of product types as basic components and by regarding strategies and tactics as the primary objects of discovery.

The mode/strategy framework encourages development of analytic models for complex, real-world conceptual construction tasks, rather than predictive models or simulation systems for simple or artificial tasks. In this respect, it is similar to the Hayes and Flower approach and differs from the GOMS and IPS approaches that emphasize generative models and simulation systems. The reasons for this are practical — we simply do not have an adequate base of knowledge at this time to support generative models that can address interesting or meaningful issues for this category of tasks. However, by developing formal analytic models and by using those models to study users' strategies and behaviors, we may eventually be able to build a sufficient base of knowledge that would make generative models of complex tasks practical. Thus, I see the two approaches as potentially complementary.

Like GOMS, the mode/strategy framework can be used to describe users' interactions with computer systems. However, unlike GOMS, it provides direct guidance for designing new or improved systems. This results from the direct and natural mapping between a set of cognitive modes and the flow of information among them to a set of system modes and the flow of data among them. Because a system built in this way has a well-defined relationship with a cognitive model of the task, we can identify the specific ways in which that system amplifies its users' cognitive skills as well as the extent of change. Thus, the mode/strategy framework makes it possible to talk about intelligence amplification in precise and meaningful ways.

Although the mode/strategy model of writing resembles the Hayes and Flower model in its emphasis on analysis, it differs from it in several important respects. First, it is defined within the terms of an explicit architecture, rather than by example in terms of a collection of specific processes. Second, because it is expressed as a computer program, all of its terms and rules are well-defined. Thus, it is a formal, executable model, as opposed to an informal conceptual model. Third, it can be applied automatically and consistently across multiple subjects, rather than manually by human judges who sometimes disagree in their interpretations.

Many of these differences are subtle, but together they add up to an approach that is particularly well-suited for handling complex, computer-based conceptual construction tasks.

Objection to Collective Intelligence

Before concluding this discussion of concepts drawn from the cognitive science literature, I briefly note an objection raised by Allen Newell to the notion of a collective intelligence, based on the rate at which knowledge can be communicated from one individual to another. Because Newell has been so influential in the development of cognitive science, this objection must be addressed; consequently, I include his argument in his own words:

> A social system, whether a small group or a large formal organization, ceases to act, even approximately, as a single rational agent. Both the knowledge and the goals are distributed and cannot be fully brought to bear in any substantial way on any particular decision. This failure is guaranteed by the very small communication bandwidth between humans compared with the large amount of knowledge available in each human's head. . . . Modeling groups as if they had a group mind is too far from the truth to be a useful scientific approximation very often. (Newell, 1990, pp. 490-491))

Elsewhere, he explained in more detail his assumptions and reasoning regarding fundamental limits in human communication:

> Let the rate of knowledge intake (or outflow) between the human and the environment be ~~K chunks/sec. Then this same rate governs the acquisition of knowledge prior to the meeting of the group that attempts to share knowledge by communication. The total body of knowledge in a group member is ~~KT, where T is the total lifetime of the member. The amount that can be shared in the group meeting is ~~KΔT, where ΔT is the duration of the group meeting. But the group meeting time, ΔT, is small compared to T, independent of K, the communication rate. . . . The net effect is that the social band becomes characterized as a distributed set of intendedly rational agents, in which each agent has a large body of knowledge relative to how fast it can communicate it. (Newell, 1990, p. 155)

Thus, Newell argued that for a group to exhibit collective intelligence, all members of the group must share the complete body of knowledge and goals relevant to the task that are held separately by its individual members. Newell is right that no group can achieve total integration of knowledge such as this. However, this may be too strong a requirement. Individuals do not always utilize all of the

potentially relevant knowledge they possess for each decision or mental operation they perform. Similarly, total shared knowledge may not be required to achieve a level of coherence within a group sufficient to justify a concept of collective intelligence. I try to speak to this objection at several points in the discussion that follows.

Summary

In this chapter, I looked at cognition as an information processing activity. The discussion began with a review of three general models or architectures developed from this perspective. A basic set of components was identified in Newell and Simon's original IPS architecture that are subsequently refined in Newell's Soar and Anderson's Act* systems. These include a long-term memory, possibly divided into separate stores for procedural and declarative knowledge; a working memory; and a separate processor component or, alternatively, a set of processes that function within working memory. These systems also included a context for higher level conceptual thought called a problem space. This context, in turn, included a goal, a data type, and a data structure as well as set of rudimentary strategies for constructing a path from an initial data state to the goal data state. The architectures differed in their respective views of a problem space as part of the basic cognitive architecture or as an informational object acquired as part of learning a particular skill.

Next, I looked at several models and frameworks that applied these ideas to specific tasks, such as writing, or specific situations, such as users working with computers to perform a task. These discussions added several new concepts. First, the process model of Hayes and Flower showed that by taking an analytic approach, one can address conceptual construction tasks that cannot currently be addressed using predictive or generative models. Second, the GOMS model provided a set of categories for describing tasks performed in close conjunction with a computer system. It is limited in the tasks it can model, however, by the fundamental role played by its selection rules component. Finally, the mode/strategy framework combines aspects of both approaches. Its practice of expressing task models as

grammars leads to formal analytic models based on a well-defined architectural construct. When these grammars are implemented as parsing programs, they can support extensive studies of users' cognitive behaviors, perhaps leading to more sophisticated concepts of strategies and tactics, currently missing from conventional IPS models.

I also reviewed an objection raised by Allen Newell to the notion that a group can function as a coherent intelligent agent. It states that for a group to function as an intelligent agent, all members would have to share complete knowledge of goals and the task domain, an impossible requirement because of bandwidth limitations.

In Part II, I draw on this research in building a concept of collective intelligence by identifying components within collaborative groups that are recognizable as extrapolations of basic IPS architectural components. I also try to build a path around Newell's objection to this concept.