PARALLELISM IN AI PROBLEM SOLVING: A CASE STUDY OF HEARSAY II

R. D. Fennell and V. R. Lesser
Department of Computer Science[1]
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

October, 1975

ABSTRACT

The Hearsay II speech-understanding system (HSII) (Lesser, *et al.*, 1974; Fennell, 1975; Erman and Lesser, 1975) is an implementation of a knowledge-based multiprocessing AI problem-solving organization. HSII is intended to represent a problem-solving organization which is applicable for implementation in a multiprocessing environment, and is, in particular, currently being implemented on the C.mmp multiprocessor system (Bell, *et al.*, 1971) at Carnegie-Mellon University. The object of this paper is to explore several of the ramifications of such a problem-solving organization by examining the mechanisms and policies underlying HSII which are necessary for supporting its organization as a multiprocessing problem-solving system. First, an abstract description of a class of problem-solving systems is given using the Production System model of Newell (1973). Then, the HSII problem-solving organization is described in terms of this model. The various decisions made during the course of design necessitated the introduction of various multiprocessing mechanisms (e.g., mechanisms for maintaining data localization and data integrity), and these mechanisms are discussed. Finally, a simulation study is presented which details the effects of actually implementing such a problem-solving organization for use in a particular application area, that of speech understanding.

[1] This research was supported in part by the Defense Advanced Research Projects Agency of the Office of the Secretary of Defense (Contract F44620-73-C-0074) and monitored by the Air Force Office of Scientific Research.

INTRODUCTION

Many AI problem-solving tasks require large amounts of processing power in order to achieve solution in any given computer implementation of a problem-solving strategy. The amount of processing power required is directly related to the size of the search space which is examined during the course of problem solution. Exhaustive search of the state space associated with almost any problem of interest is precluded due to the sheer size of the state space.[1] In most problem-solving attempts, heuristics are employed which prune the search space to a more manageable size. However, searching even the reduced state space often requires large amounts of processing power. The demand for sufficient computing power becomes critical in tasks requiring real-time solution, as is the case in the speech-understanding task with which this paper is primarily concerned.. For example, a speech-understanding system capable of reliably understanding connected speech involving a large vocabulary and spoken by multiple speakers is likely to require from 10 to 100 million instructions per second of computing power, if the recognition is to performed in real time.[2] Recent trends in technology suggest that this computing power can be economically obtained through a closely-coupled network of asynchronous "simple" processors (involving perhaps 10 to 100 of these processors), (Bell, *et al.*, 1973, and Heart, *et al.*, 1973). The major problem (from the problem-solving point of view) with this network multiprocessor approach for generating computing power is in devising the various problem-solving algorithms in such a way as to exhibit a structure appropriate for exploiting the parallelism available in the multiprocessor network, for it is only by taking advantage of this processing parallelism that the desired effective computing power will be achieved.

The Hearsay II speech-understanding system (HSII) (Lesser, *et al.* 1974; Fennell, 1975; and Erman and Lesser, 1975) currently under development at Carnegie-Mellon University represents a problem-solving organization that can effectively exploit a multiprocessor system. HSII has been designed as an AI system organization suitable for expressing *knowledge-based problem-solving strategies* in which appropriately

[1] As an example, consider the chess-playing task. In an end game situation, there are typically 20 legal moves at each ply (half-move); so for a search depth of 6 plies, the search space will have 64 million branches.

[2] The Hearsay I (Reddy, *et al.*, 1973a,b,c and Erman, 1974) and Dragon (Baker, 1975) speech understanding systems require approximately 10 to 20 mips of computing power for real-time recognition when handling small vocabularies.

organized subject-matter knowledge may be represented as *knowledge sources* capable of contributing their knowledge in a parallel data-directed fashion. A *knowledge source* may be described as an agent that embodies the knowledge of a particular aspect of a problem domain and is useful in solving a problem from that domain by performing actions based upon its knowledge so as to further the progress of the overall solution. It is felt that the knowledge source is an appropriate unit for use in the decomposition of a knowledge-intensive task domain. Knowledge sources, being suitably organized capsules of subject-matter knowledge, may be independently formulated as various pieces of the knowledge relevant to a task domain become crystallized. The HSII system organization allows these various independent and diverse sources of knowledge to be specified and their interactions coordinated so they might cooperate with one another (perhaps asynchronously and in parallel) to effect a problem solution. As an example of the decomposition of a task domain into knowledge sources, in the speech task domain there might be distinct knowledge sources to deal with acoustic, phonetic, lexical, syntactic, and semantic information. While the speech task is the first test of the multiprocessing problem-solving organization of HSII, it is believed that the system organization provided by HSII is capable of expressing other knowledge-based AI problem-solving strategies, as might be found in vision, robotics, chess, natural language understanding, and protocol analysis. In fact, proposals are under way which will further test the applicability of HSII by implementing a system for the analysis of natural scenes using the HSII problem-solving organization (Ohlander, 1975).

The rest of this paper will explore several of the ramifications of such a problem-solving organization by examining the mechanisms and policies underlying HSII which are necessary for supporting its organization as a multiprocessing problem-solving system. First, an abstract description of a class of problem-solving systems is given using the Production System model of Newell (1973). Then, the HSII problem-solving organization is described in terms of this model. The various decisions made during the course of design necessitated the introduction of various multiprocessing mechanisms (e.g., mechanisms for maintaining data localization and data integrity), and these mechanisms are discussed. Finally, a simulation study is presented which details the effects of actually implementing such a problem-solving organization in a multiprocessor environment.

THE MODEL

An Abstract Model for Problem Solving

In the abstract, the problem-solving organization underlying HSII may be modeled in terms of a "production system," (Newell, 1973). A ***production system*** is a scheme for specifying an information processing system in which the control structure of the system is defined by operations on a set of *productions* of the form 'P → A', which operate from and on a collection of data structures. 'P' represents a logical antecedent, called a *precondition*, which may or may not be satisfied by the information encoded within the dynamically current set of data structures. If 'P' is found to be satisfied by some data structure, then the associated *action* 'A' may be executed, which presumably will have some altering effect upon the data base such that some other (or the same) precondition becomes satisfied. This paradigm for sequencing of the actions can be thought of as a data-directed control structure, since the satisfaction of the precondition is dependent upon the dynamic state of the data structure. Productions are executed as long as their antecedent preconditions are satisfied, and the process halts either when no precondition is found to be satisfied or when an action executes a stop operation (thereby signalling problem solution or failure, in the case of problem-solving systems).

The HSII Problem-Solving Organization: A Production System Approach

The HSII system organization, which can be characterized as a "parallel" production system, has a centralized data base which represents the dynamic problem solution state. This data base, which is known as the ***blackboard***, is a multidimensional data structure which is readable and writable by any precondition or knowledge-source process (where a knowledge-source process is the embodiment of a production action).[1] Preconditions are procedurally oriented and may specify arbitrarily complex tests to be performed on the data structure in order to decide precondition satisfaction.

[1] As an example, the dimensions of the HSII speech-understanding system data base are informational level (e.g., acoustic level, phonetic level, and word level), utterance time (speech time measured from the beginning of the input utterance), and data alternatives (where multiple hypotheses are permitted to exist simultaneously at the same level and utterance time). For additional details, see Appendix A.

Preconditions are themselves data-directed in that they are tested for satisfaction whenever relevant changes occur in the data base;[1] and simultaneous precondition satisfaction is permitted. Testing for precondition satisfaction is not presumed to be an instantaneous or even an indivisible operation, and several such precondition tests may proceed concurrently.

The knowledge-source processes representing the production actions are also procedurally oriented and may specify arbitrarily complex sequences of operations to be performed upon the data structure. The overall effect of any given knowledge-source process is usually either to hypothesize new data which is to be added to the data base or to verify (and perhaps modify) data previously placed in the data base. This follows the general *hypothesize-and-test* problem-solving paradigm wherein hypotheses representing partial problem solutions are generated and then tested for validity; this cycle continues until the verification phase certifies the completion of processing (and either the problem is solved or failure is indicated). The execution of a knowledge-source process is usually temporally disjoint from the satisfaction of its precondition; the execution of any given knowledge-source process is not presumed to be indivisible; and the concurrent execution of multiple knowledge-source processes is permitted. In addition, a precondition process may invoke multiple instantiations of a knowledge source to work on the different parts of the blackboard which independently satisfy the precondition's pattern. Thus, the independent data-directed nature of precondition evaluation and knowledge-source execution can potentially generate a significant amount of parallel activity through the concurrent execution of different preconditions, different knowledge sources, and multiple instantiations of a single knowledge source.

[1] In effect, preconditions themselves have preconditions, call them "pre-preconditions." In HSII, knowledge-source preconditions (which correspond to action preconditions in the production system model) may be arbitrarily complex. In order to avoid executing these precondition tests unnecessarily often, they in turn have pre-preconditions which are essentially monitors on relevant primitive data base events (e.g., monitoring for a change to a given field of a given node in the data base, or a given field of any node in the data base). Whenever any of these primitive events occurs, those preconditions monitoring such events are awakened and allowed to test for full precondition satisfaction. These data events are used by the precondition process as pointers to the specific parts of the data base which may now satisfy the pattern the precondition is monitoring for. During the period between when the precondition process has been first awakened and the time it is executed, the monitoring for relevant data base events continues. Thus, a precondition process, when finally executed, may check more than one part of the data base for satisfaction.

The basic structure and components of the HSII organization may be depicted as shown in the message transaction diagram of Figure 1. The diagram indicates the paths of active information flow between the various components of the problem-solving system as solid arrows; paths indicating control activity are shown as broken arrows. The major components of the diagram include a passive global data structure (the *blackboard*) which contains the current state of the problem solution. Access to the blackboard is conceptually centralized in the *blackboard handler* module, [1] whose primary function is to accept and honor requests from the active processing elements to read and write parts of the blackboard. The active processing elements which pose these data access requests consist of *knowledge-source processes* and their associated *preconditions*. Preconditions are activated by a *blackboard monitoring mechanism* which monitors the various write-actions of the blackboard handler; whenever an event occurs which is of interest to a particular precondition process, that precondition is activated. If upon further examination of the blackboard, the precondition finds itself "satisfied," the precondition may then request a process instantiation of its associated knowledge source to be established, passing the details of how the precondition was satisfied as parameters to this instantiation of the knowledge source. Once instantiated, the knowledge-source process can respond to the blackboard data condition which was detected by its precondition, possibly requesting further modifications be made to the blackboard, perhaps thereby triggering further preconditions to respond to the latest modifications. This particular characterization of the HSII organization, while certainly overly simplified, shows the data-driven nature of the knowledge source activations and interactions.

The following sections of this paper will attempt to refine this diagram of the HSII organization by pointing out the difficulties that arise from this oversimplified representation of the organization and by supplementing the various components of this simple diagram to resolve these problems and result in a more complete organization for AI problem-solving in multiprocessing environments. A more complete message transaction diagram for HSII will be presented in a subsequent section.

[1] The blackboard handler module could be implemented either as a procedure which is called as a subroutine from precondition and knowledge source processes, or as a process which contains a queue of requests for blackboard access and modification sent by precondition and knowledge source processes. In the implementation discussed in this paper, the blackboard handler module is implemented as a subroutine.

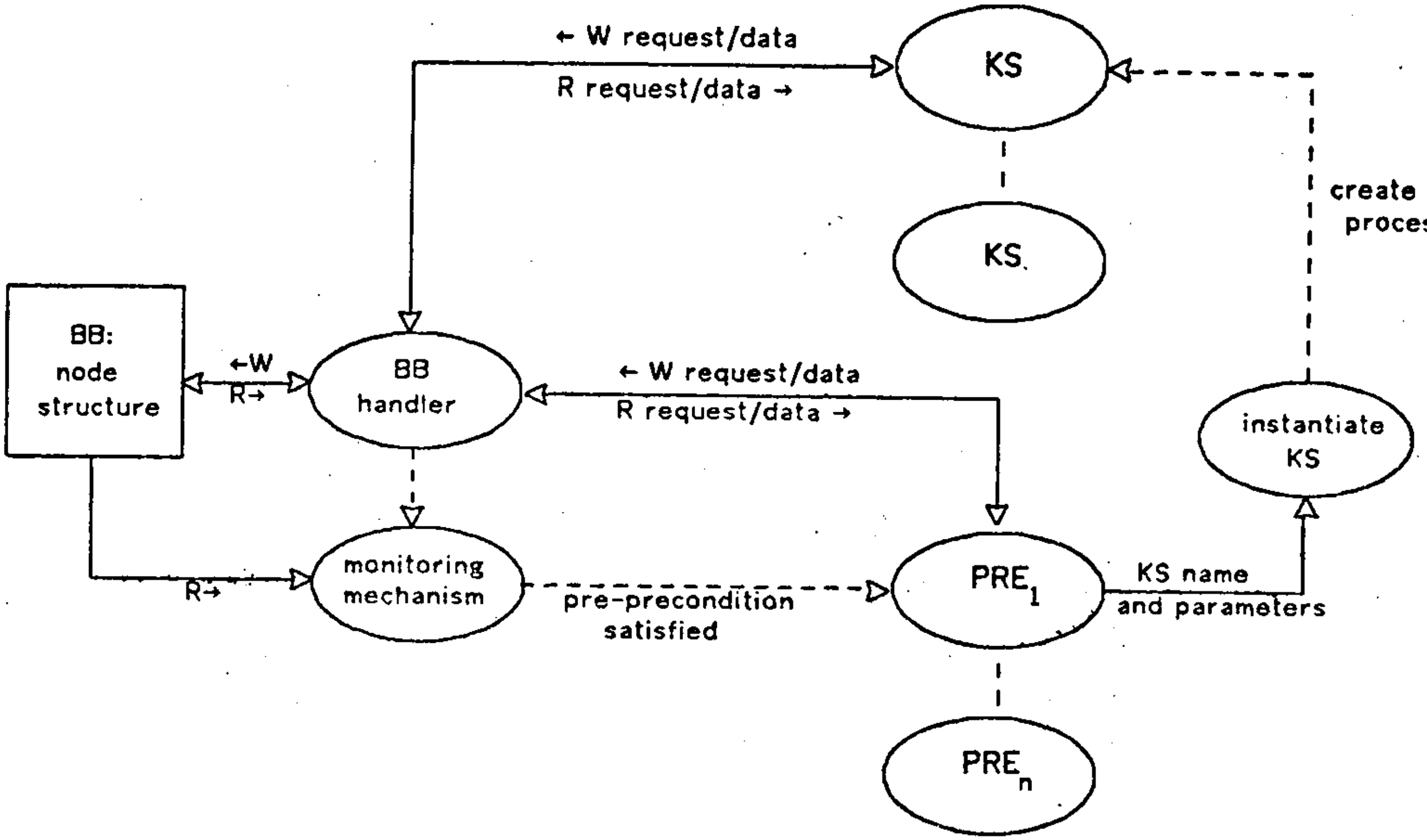

Figure 1. Simplified HSII System Organization

HEARSAY II MULTIPROCESSING MECHANISMS

Given the decision that multiple preconditions may be simultaneously satisfied and that multiple knowledge-source processes may execute concurrently, various mechanisms must be provided to accommodate such a multiprocessing environment. Mechanisms must be provided to support the individual localized executions of the various active and ready processes and to keep the processes from interfering with one another, either directly or indirectly. On the other hand, mechanisms must also be provided so that the various active processes may communicate with one another so as to achieve the desired process cooperation. Since the various constituent knowledge sources are assumed to be independently developed and are not to presume the explicit existence of other knowledge sources, communication among these knowledge sources must necessarily be indirect. The desire for a modular knowledge source structure arises from the fact that usually many different people are involved in the implementation of the set of knowledge sources, and, for purposes of experimentation and knowledge source performance analysis, the system should be able to be easily reconfigured with alternative subsets of knowledge sources. This communication takes two primary forms: data base monitoring for collecting pertinent data event information for future use (*local contexts* and precondition activation), and data base monitoring for the occurrence of data events which violate prior data assumptions (*tags* and *messages*). The following paragraphs will discuss these forms of data base monitoring and their relationship to the data access synchronization mechanisms required in a multiprocess system organization.

Local Contexts

Interprocess communication (and interference) among knowledge sources and their associated preconditions occurs mainly via the global data base, as a result of the design decisions involved in trying to maintain process independence. It is therefore not surprising that the mechanisms necessary to bring about the desired process cooperation and independence are based on global data base considerations. The global data base (the *blackboard*) is intended to contain only dynamically current information. Since preconditions (being data-directed) are to be tested for satisfaction upon the occurrence of relevant data base changes (which are historical *data events*), and since neither precondition testing nor action execution (nor the sequential combination of the two) is assumed to be an indivisible operation, localized data bases must be provided for each process unit (precondition or action) which needs to remember relevant historical data events. These localized databases, called *local contexts* in HSII, which record the changes to the blackboard since the precondition process was last executed or the knowledge source process was created provide personalized operating environments for the various precondition and knowledge-source processes. A local context preserves only those data events[1] and state changes relevant to its owner. The creation time of the local context (i.e., the time from which it begins collecting data events) is also dependent upon the context owner. Any given local context is built up incrementally: when a modification occurs to the global data base, the resulting data event is distributed to the various local contexts interested in such events. The various primitive data modification routines (or node creation routines) are responsible for the distribution of the data events which result from the modification, just as these modification routines are also responsible for sending warning messages to those processes which want to be notified when specific characteristics of a particular node are altered.[2] Thus, the various local contexts retain a history of relevant data events, while the global data base contains only the most current information.

Data Integrity

Since precondition and knowledge-source processes are not guaranteed to be executed uninterruptedly, these processes often need to assure the integrity of various assumptions they are making about the contents of the data base; for should these assumptions become violated due to the actions of an intervening process, the further computation of the assuming process may have to be altered (or terminated). One way to approach the problem of data integrity is to guarantee the validity of data assumptions by disallowing intervening processes the ability to modify (or perhaps even to examine) critical data. In order to guarantee the integrity of data through the mechanism of exclusive access, the HSII system provides two forms of locking primitives, *node-* and *region-locking*. Node-locking guarantees exclusive access to an explicitly

[1] The information which defines a data event consists of the locus of the event (i.e., a data node name and a field name within that node) and the old value of the field (the new value being stored in the global data base).

[2] The use of these warning messages as way of preserving data integrity will be discussed in the next section.

specified node in the blackboard, whereas region-locking guarantees exclusive access to a collection of nodes that are specified implicitly based on a set of node characteristics. In the current implementation of HSII, the region characteristics are specified by a particular information level and time period of a node. If the blackboard is considered as a two-dimensional structure with coordinates of information-level and time, then region-locking permits the locking of an arbitrary rectangular area in the blackboard. Region-locking has the additional property of preventing the creation of any new node that would be placed in the blackboard area specified by the region by other than the process which had requested the region-lock. Additional locking flexibility is introduced by allowing processes to request read-only access to data fields; this reduces possible contention by permitting multiple readers of a given field to coexist, while excluding any writers of that field until all readers are finished. The system also provides a "super lock," which allows an arbitrary group of nodes and regions to be locked at the same time. A predefined linear ordering strategy for non-preemptive data access allocation (Coffman, *et al.*, 1971) is applied by the "super lock" primitive to the desired node- and region-locks so as to avoid the possibility of data base deadlock.

However, this technique of guaranteeing data integrity through exclusive access is only applicable if all the nodes and regions to be accessed and modified are known ahead of time. The sequential acquisition of exclusive access to nodes and region, without intervening unlocks, can result in the possibility of deadlock. In the HSII blackboard, nodes are interconnected to form a directed graph structure; because it is possible to establish an arbitrarily complex interconnection structure, it is often very difficult for a knowledge-source process to anticipate the sequence of nodes it will desire to access or modify. Thus, the mechanisms of exclusive access cannot always be used to guarantee data integrity in a system with a complex data structure and a set of unknown processes. Further, even if the knowledge source can anticipate the area in the blackboard within which it will work and thereby request exclusive access to this area, the area may be very large, thus leading to a significant decrease in potential parallel activity caused by other processes waiting for this locked area to become available.

An alternative approach to guaranteeing data integrity is to provide a means by which a process (precondition or knowledge source) may place data assumptions about the particular state of a node or group of nodes in the data base (the action of putting these assumptions in the blackboard is called *tagging*). If these assumptions are invalidated by a subsequent blackboard modification operation of another process, then a *message* indicating this violation is sent to the process making the assumption. In the meantime, the assuming process can proceed without obstructing other processes, until such time as it intends to modify the data base (since data base modification is the only way one process can affect the execution of another). The process must then acquire exclusive access to the parts of the data base involved in its prior assumptions (which parts will have been previously *tagged* in the data base to define a *critical data set*)[1] and check to see whether the assumptions have been violated (in which case, messages indicating those violations would have been sent to the process). If a violation has occurred, the assuming process may wish to take alternative action; otherwise, the intended data base modifications may be made as if the process had had exclusive access throughout its computation. This tagging mechanism can also be used to signal the knowledge-source process that the initial conditions in the blackboard (i.e., the precondition pattern) that caused the precondition to invoke it have been modified; this is accomplished by having the precondition tag these initial conditions on behalf of the knowledge-source process prior to the instantiation of the knowledge source.

In summary, the HSII organization provides mechanisms to accomplish both of these forms of data integrity assurance: the various data base locking mechanisms described previously provide several forms of exclusive or read-only data access; and the data tagging facility allows data assumptions to be placed in the data base without interfering with any process' ability to access or modify that area of the data base (with data invalidation warning messages being sent by data base monitors whenever the assumptions are violated).

To provide a basis for the discussion in the subsequent sections of this paper, Figure 2, depicting the various components of the HSII organizational structure, is offered. The diagram is a more detailed version of the message transaction model presented previously. The new components of this diagram are primarily a result of addressing multiprocessing considerations.

As in the earlier, more simplified organizational diagram, the dynamically current state of the problem solution is contained in a centralized, shared data base, called the *blackboard*. The blackboard not only contains data *nodes*, but it also records

[1] Actually, the requirement is that no other process be able to write to these parts of the data base.

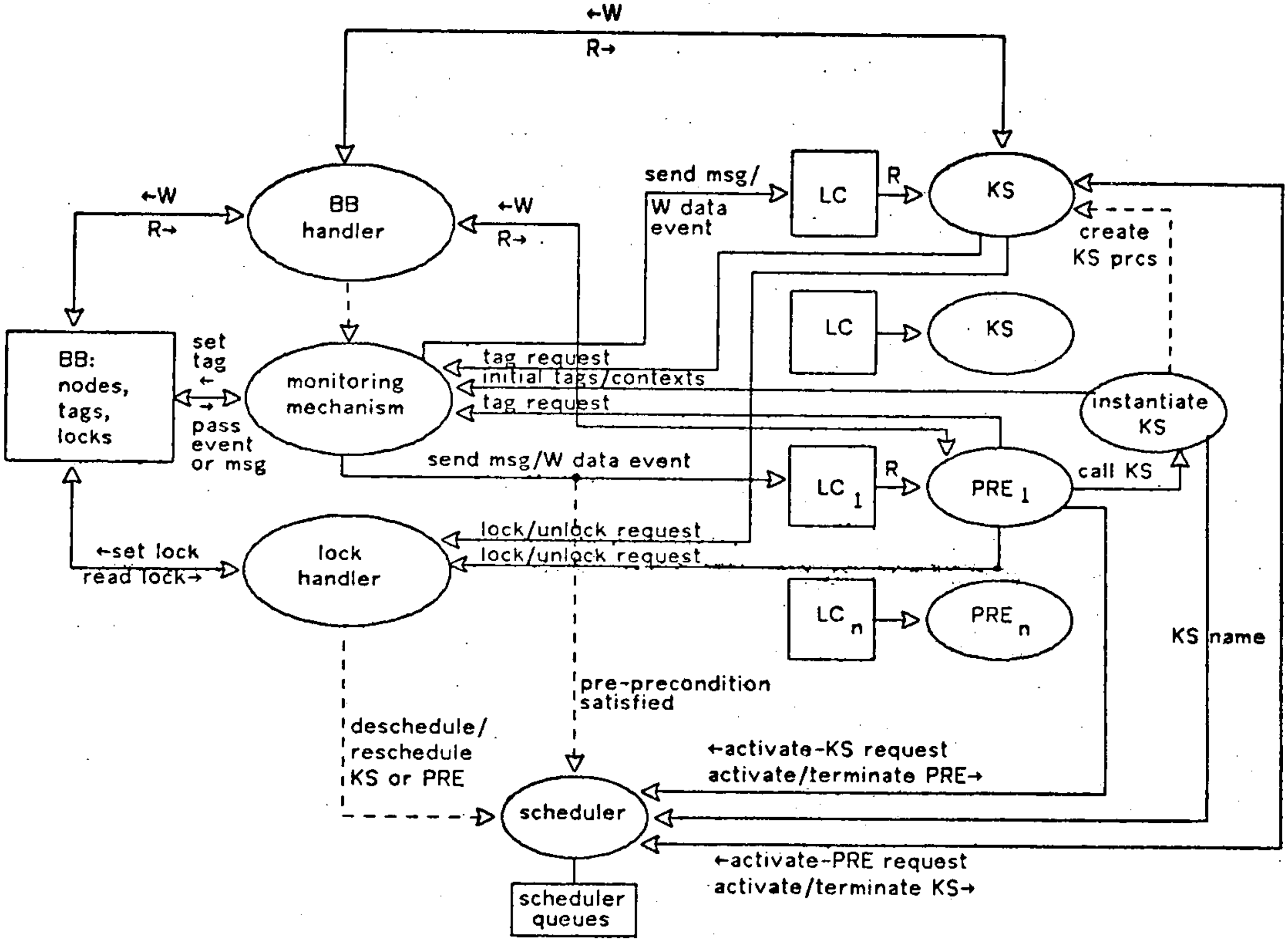

Figure 2. HSII System Organization

data monitoring information (*tags*) and data access synchronization information (*locks*). Access to the blackboard is conceptually centralized in three modules. As before, the *blackboard handler* module accepts and honors read and write data-access requests from the active processing elements (the *knowledge-source processes* and their *precondition processes*). A *lock handler* coordinates data-access synchronization requests from the knowledge-source processes and preconditions, with the ability to block the progress of the requesting process until the synchronization request may be satisfied. A *monitoring mechanism* is responsible for accepting *data tagging* requests from the knowledge-source processes and preconditions, and for sending *messages* to the tagging processes whenever a tagged data field is modified. It is also the responsibility of the monitoring mechanism to distribute *data events* to the various *local contexts* of the knowledge-source processes and preconditions, as well as to activate precondition processes whenever sufficient data events of interest to those preconditions have occurred in the blackboard.

Associated with each active processing element is a local data base, the *local context*, which records data events that have occurred in the blackboard and are of interest to that particular process. The local contexts may be read by their associated processes in order to find out which data nodes have been modified recently and what the previous values of particular data fields were. The local contexts are automatically maintained by the blackboard monitoring mechanism.

Upon being activated and satisfied, precondition processes may instantiate a knowledge source (thereby creating a *knowledge-source process*), passing along the reasons for this instantiation as parameters to the new knowledge-source process and at the same time establishing the appropriate data monitoring connections necessary for the new process. The *goal-directed scheduler* retains the actual control over allocating hardware processing capability to those knowledge-source processes and precondition processes which can best serve to promote the progress of the problem solution.[1]

[1] One way a scheduler might help in reducing (or eliminating) global data base access interference is to schedule to run concurrently only processes whose global data demands are disjoint. Such a scheduling policy could even be used to supplant an explicit locking scheme, since the global data base locking would be effectively handled by the scheduler (albeit probably on a fairly gross level). Of course, other factors may rule out such an approach to data access synchronization, such as an inability to make maximal use of the available processing resources if only data-disjoint processes are permitted to run concurrently, or the inability to know in

EXPERIMENTS WITH AN IMPLEMENTATION

The preceding sections of this paper have presented various of the mechanisms necessary in implementing a knowledge-based problem-solving system such as HSII in a multiprocessing environment. The present sections will discuss the various experiments that have been performed in an attempt to characterize the multiprocessing performance of the HSII organization in the speech-understanding task.

HSII Multiprocess Performance Analysis through Simulation

In order to gain insight into the various efficiency issues involving multiprocess problem-solving organizations, a simulation model was incorporated within the uniprocessor version of the HSII speech-understanding system. The HSII problem-solving organization was not itself modeled and simulated, but rather the actual HSII implementation (which is a multiprocessing organization even when executing on a uniprocessor) was modified to permit the simulation of a hardware multiprocessor environment.

There were four primary objectives of the simulation experiments: a) to measure the software overheads involved in the design and execution of a complicated, data-directed multiprocess(or) control structure, b) to determine whether there really exists a significant amount of parallel activity in the speech-understanding task, c) to understand how the various forms of interprocess communication and interference, especially that from data access synchronization in the blackboard, affect the amount of effective parallelism realized, and d) to gain insight into the design of an appropriate scheduling algorithm for a multiprocess problem-solving structure. Certainly, any results presented will reflect the detailed efficiencies and inefficiencies of the particular system implementation being measured, but hopefully the organization of HSII is sufficiently general that the various statements will have a wider quantitative applicability for those considering similar multiprocess control structures.

By way of summary, the primary characteristics of the HSII organization

advance the precise blackboard demands of each knowledge-source instantiation. Nonetheless, the information relating to the locality of knowledge-source data references is useful in scheduling processes so as to avoid excessive data access interference (thereby improving the effective parallelism of the system).

include: a) multiple, diverse, independent and asynchronously executing knowledge sources, b) cooperating (in terms of control) via a generalized form of the hypothesize-and-test paradigm involving the data-directed invocation of knowledge-source processes, and c) communicating (in terms of data) via a shared blackboard-like data base in which the current data state is held in a homogeneous, multidimensional, directed-graph data structure.

The HSII Speech Understanding System: The Simulation Configuration

The configuration of the HSII speech-understanding system, upon which the following simulation results were based, consists of eight separate generic knowledge sources (each of which may be realized by several active instantiations at any given moment during the problem solution), each of which represents some body of knowledge relevant to the speech-understanding task. Due to the excessive cost of the simulation effort (and due to the limited stages of development of some available knowledge sources), only a subset of the available knowledge sources was actually used in the simulation experiments. Appendix A (which was extracted from (Lesser, *et al.*, 1974)) contains a more detailed description of the blackboard and the various knowledge sources for the more complete HSII speech-understanding system. The knowledge sources used in the simulation were: the *Segment Classifier*, the *Phone Synthesizer* (consisting of two knowledge sources), the *Phoneme Hypothesizer*, the *Phone-Phoneme Synchronizer* (consisting of three knowledge sources), and the *Rating Policy Module*. These knowledge sources are activated by half a dozen precondition processes (which are permanently instantiated in the system), which are continuously monitoring the blackboard data base for events and data patterns relevant to their associated knowledge sources. Both knowledge sources and preconditions may freely access the centralized blackboard data base, which consists of nine lexicon levels.[1] The particular levels used were chosen so as to facilitate the information exchange between the various component knowledge sources.

This set of knowledge sources and preconditions and the associated operating system facilities provided by the HSII organization were first implemented to execute on

[1] While there are eight conceptual information levels within the HSII speech-understanding system (see Appendix A), the blackboard is abstractly segmented according to *lexicons*, rather than information levels, since lexicons allow a finer abstract decomposition of the blackboard.

a uniprocessor DECsystem-10 computer. The particular implementation represented here was programmed in the Algol-like language, SAIL (Swinehart and Sproull, 1971), using SAIL's multiprocessing facilities (Feldman, *et al.*, 1972) and making extensive use of its LEAP associative data storage facility (Feldman and Rovner, 1969). Thus, while the hardware environment of this version of the HSII speech-understanding system is that of a single processor, the software environment is the multiprocessing structure described throughout this paper. The simulation experiments were then run using this HSII configuration, simulating the hardware environment of a closely-coupled multiprocessor where processors can directly communicate with each other through shared memory. The size of the HSII configuration used in the simulations was about 180K, 36-bit words; 70K of this total was the HSII operating system plus the SAIL runtime routines, 73K was precondition and knowledge source code plus variables, and the remainder (which varied from 20K to 45K depending on the number of processors being simulated and the number of processes being instantiated) represented the blackboard data base plus process activation records and other SAIL working space. The simulations were carried out to determine the efficiencies of the various HSII multiprocessing mechanisms discussed previously, as well as to gain some insight into any problems that might arise in the ensuing implementation of a HSII speech-understanding system for the Carnegie-Mellon C.mmp multiprocessor.[1] The following sections will discuss the results of the various experiments which have been performed using the multiprocessor-simulation version of the HSII speech-understanding system.

Simulation Mechanisms and Simulation Experiments

The various multiprocessor simulation results were obtained by modifying the flow of control through the usual HSII multiprocessing organization to allow simulation scheduling points every time a running process could interact in any way with some other concurrently executing process. Such points included blackboard data base accesses and data base access synchronization points (including attempts to acquire data base resources, both at the system and user levels, and any resulting points of

[1] The implementation of the C.mmp version of the HSII speech-understanding system thus far has been, in fact, essentially a direct mapping of the DECsystem-10 implementation, with additional design being done as necessary to solve the particular problems of running in the C.mmp environment (such as having to resolve the small address space problem, wherein any given process may have at any one moment only a 32K-word window into the centrally located main memory).

process suspension due to the unavailability of the requested resource, as well as the subsequent points of process wake-up for retrying the access request). Simulation scheduling points were also inserted whenever a data modification warning message (triggered by modifying a tagged data field) was to be sent, as well as whenever a process attempted to receive such a message. The scheduling mechanism itself was also modified to allow for the simulated scheduling of multiple processing units, while maintaining the state information associated with each processor being simulated (such as the processor clock time of that simulated processor and the state of the particular process being run on that processor). The simulation runs were performed so as to keep the processor clock-times of each processor being simulated in step with one another (the simulation being *event-driven*, rather than *sampled*), thereby allowing for the accurate measurement and comparison of concurrent events across processors. By selecting the number of processors to be simulated and choosing the usual scheduling parameters and precondition and knowledge-source parameters, a chronological trace of the activity of each process and processor could be obtained. By accumulating statistics during the trace period and by performing various post-processing operations upon this activity trace record, the simulation results presented in the following sections were obtained.

Most of the results presented here were achieved by using a single set of knowledge sources (as described above), with a single speech-data input utterance, keeping the data base locking structure and scheduling algorithms essentially fixed, while varying the number of simulated (identical) processors. Several runs were also performed to test the effects of altering the knowledge-source set, altering the locking structure, and altering the mode of data input (the normal input mode being a utterance-time-ordered introduction of input data which simulates real-time speech input).

Measures of Multiprocessing Overhead: Primitive Operation Timings

Time measurements of various primitive operations were made using a 10-microsecond hardware interval timer. Some of the timed primitive operations (such as those involving simple data base access and modification) were not especially subject to the fact that the problem-solving organization involved multiple parallel processes, whereas others (such as those involving process instantiation and process synchronization) were directly related to the multiprocess aspects of the organization (and might even be taken in part as overhead when compared to alternative single-process system organizations). The times for the various system operations, as shown in Table 1, should be read as relative values, comparing the multiprocess-oriented operations with the data accessing operations to get a relative feel for the overheads involved in supporting and maintaining the multiprocess organization of HSII. Keep in mind that such time measurements are highly dependent on the particular implementation and can change fairly radically when implemented differently. In fact, a primary use of such timings is in determining operating system bottlenecks so that such code sections can be rewritten in a more optimal way. As a result, some primitive operations reflect execution times which are a result of extensive optimization attempts, while other operations (in particular, the "super lock" operations, *lock!* and *unlock!*) have not yet been subjected to this optimization.

Table 1 gives timing statistics relating to the costs involved in maintaining the shared, centralized blackboard data base. Two sets of statistics are given, one set showing the operation times without the influence of data access synchronization (blackboard locking) and one set with the locking structures in effect. These two sets of times give a quantitative feeling for the cost of data access synchronization mechanisms in this particular implementation of HSII. The figures given include the average runtime cost per operation, the number of calls (in this particular timing run) to each operation (thereby showing the relative frequencies of operation usage), and the percentage of the overall runtime consumed by each operation. With respect to the individual entries, *create.node* is a composite operation (involving many field-writes and various local context updates) for creating blackboard nodes. The *read.node.field* and *write.node.field* operations are used in accessing the individual fields of a node. Note that included in any given field-read or -write operation is the cost of perhaps tagging (or untagging) that particular field (or its node). The various functions of the blackboard monitoring mechanism are contained within the field-write operations. Thus, also included in the field-write operation is the cost of distributing the data event resulting from the write operation to all relevant precondition and knowledge-source process local contexts, as well as the cost of sending tag messages to all processes which may have tagged the field being modified; these additional costs are also accounted for independently in the *send.msgs.and.events* and *notify.sset* table entries. Field-write operations are also responsible for evaluating any pre-preconditions associated with the field being modified and activating any precondition whose pre-

	% total runtime		mean time (ms)		number of calls	
	w/o	w/	w/o	w/	w/o	w/
	lock	lock	lock	lock	lock	lock
Blackboard Accessing:						
create.node	6.96	4.15	35.81	50.77	287	287
read.node.field	5.06	15.68	0.31	2.03	23577	25279
write.node.field	14.13	7.75	13.96	18.44	1493	1476
Blackboard Associative Retrieval:						
retrieve	2.72	4.98	25.07	109.45	160	160
get.time.adjacent	9.31	15.33	23.44	92.00	586	586
get.struct.adjacent	3.99	6.31	43.35	163.20	136	136
get.nodes.in.rgn	2.05	0.87	2.98	3.00	1015	1015
Process Handling:						
invoke.ks	5.29	2.30	22.64	23.64	345	342
create.ks.prcs	0.75	0.31	3.21	3.22	345	342
ks.cleanup	8.20	5.24	35.06	53.94	345	342
invoke.pre	0.10	1.04	10.44	10.59	14	14
create.pre.prcs	0.42	0.40	8.53	19.57	72	72
Local Context Maintenance:						
transfer.tags	7.12	2.99	9.12	9.17	1152	1149
delete.all.tags	0.52	0.22	2.01	2.03	383	380
notify.sset	6.52	3.01	2.63	2.92	3665	3626
send.msgs.and.events	4.04	2.12	3.68	4.68	1021	1594
receive.msg	0.36	0.15	1.00	1.01	531	530
read.cset.or.sset	0.11	0.05	0.84	0.84	192	192
Data Access Synchronization:						
lock! (overhead)	---	7.78	---	57.47	---	476
unlock! (overhead)	---	3.22	---	23.78	---	476
lock.node	---	2.32	---	2.94	---	2770
exam.node	---	9.34	---	2.40	---	13675
lock.rgn	---	0.11	---	1.77	---	227
write.access.chk	---	0.41	---	0.98	---	1470
read.access.chk	---	14.45	---	1.60	---	31761

Table 1. Primitive Operation Times

precondition is satisfied. Included in the cost of reading a data field (e.g., *read.node.field*) is the cost of verifying the access right of the calling process to the node being read (which could involve a temporary-locking operation,[1] the cost of which is also given independently in the *lock.node* table entry); this access-right checking cost is also separately accounted for by the *read.access.chk* operation. It should be noted that because most of the mechanisms required to implement a data-directed control structure are embedded in the blackboard write operations, the time to execute a write operation is significantly more expensive than a read operation. However, the actual cost in terms of total run time of implementing a data-directed control structure is comparatively small in the HSII speech-understanding system, because the frequency of read operations is much higher than that of write operations. If this relative frequency for read and write operations holds for other task domains (e.g., vision, robotics), then a data-directed control structure (which is a very general and modular type of sequencing paradigm) seems to be a very reasonable framework within which to implement such tasks.

Additional blackboard operation costs are described in the Associative Retrieval section of Table 1. Associative retrieval is based on specifying partial node descriptions (called *matching prototypes*) which serve as a means of retrieving the set of blackboard nodes fitting that partial description. *Retrieve* represents the various retrieval operations possible using these matching prototypes. Retrieval from the blackboard may also be done by requesting the nodes which are time-adjacent (according to the utterance-time dimension of the speech-understanding blackboard) or structurally adjacent (according to the blackboard graph structure) to a given node (or set of nodes); *get.time.adjacent* and *get.struct.adjacent* perform these operations. Furthermore, retrieval may be done by requesting the set of nodes contained within a certain region of the blackboard (by *get.nodes.in.rgn*).

Table 1 also relates the costs of process handling within HSII. Process invocation and process creation are separated (the former being a request from a precondition or knowledge-source process to the scheduler to perform the latter), and the costs are accounted separately, as in *invoke.ks* and *create.ks.prcs*. *Ks.cleanup* is the

[1] If a process has not previously locked the node to which it desires access and the process does not have any other node locked, then the system will temporarily lock the node for the duration of the single read or write operation, without the process having explicitly to request access to the node.

cost of terminating a knowledge-source process; preconditions never get terminated. The cost of initializing and terminating a knowledge-source process (i.e., *invoke.ks* and *ks.cleanup*) is due to the overheads involved in maintaining local contexts, locking structures, and data base monitoring (tagging), all of which are necessitated by the multiprocess nature of the HSII organization. However, in a relative sense, this is not expensive, since the total overhead associated with process handling amounts to only about 9% of the overall execution time.

Additionally, local context maintenance costs are given in Table 1, since they are also a cost of having asynchronous parallel processes. While individual tag creation and deletion is handled by the primitive field-read and -write operations, tags may be transferred from a precondition to the knowledge source it has invoked via *transfer.tags* and destroyed at termination of a process via *delete.all.tags*. As noted above, *notify.sset* and *send.msg.and.events* are sub-operations of the field-write operations and represent the cost of distributing data event notifications to all relevant local contexts. *Receive.msg* is the operation used by precondition or knowledge-source processes to receive a tagging message (or perhaps wait for one, if one does not yet exist); and *read.cset.or.sset* is the operation for retrieving the information from a local context.

Finally, Table 1 gives the costs associated with the data access synchronization mechanism. *Lock!* and *unlock!* represent the overhead costs of locking and unlocking a group of nodes specified by the process requesting access rights. These two operations are among the most complex routines in the HSII operating system, the complexity arising from having to coordinate the allocation of data base resources by two independent access allocation schemes (node-locking and region-locking). This coordination is necessary in order to avoid any possibility of data base deadlock by maintaining a homogeneous linear ordering among all data resources (nodes and regions). The costs of *lock!* and *unlock!* do not include the time spent in performing the actual primitive locking operations. The primitive lock costs are given by *lock.node* (lock a node for exclusive access), *exam.node* (lock a node for read-only access), and *lock.rgn* (lock a region for exclusive access). The access-checking operations (*write.access.chk* and *read.access.chk*) are used by the blackboard accessing routines discussed above.

These timing statistics can be used to determine the amount of system overhead incurred in running precondition and knowledge-source processes under the

HSII operating system. The following summary statistics are offered, given as percentages of the total execution time, the percentages being calculated so as to avoid overlapping between categories (as, for example, factoring blackboard reading costs out of blackboard access synchronization):

Blackboard reading	16%
Blackboard writing	4%
Associative retrieval	7%
Internal computations of processes	27%
Local context maintenance	10%
Blackboard access synchronization	27%
Process handling	9%

Another way of viewing these figures is that approximately half of the execution time involves multiprocessor overheads (i.e., local context maintenance, blackboard access synchronization, and process handling). Based on the assumption that this multiprocess overhead is independent of the parallelism factor achieved,[1] then a parallelism factor of 2 or greater is required in order to recover the multiprocess overhead.

Effective Parallelism and Processor Utilization

The problem-solving organization underlying HSII was designed to take maximum advantage of any separability of the processing or data components available within that organization. Knowledge sources were intended to be largely independent and capable of asynchronous execution in the form of knowledge-source processes. Overall system control was to be distributed and primarily data-directed, being based on events occurring in a globally shared blackboard data base. The intercommunication (and interdependence) of the various knowledge-source processes was to be minimized by making the blackboard data base the primary means of communication, thereby exhibiting an indirection with respect to communication similar to the indirect data-directed form of process control. Such a problem-solving organization was believed to be particularly amenable to implementation in the hardware environment of a network of closely-coupled asynchronous processors which share a common memory. Given

[1] This assumption, based on timing statistics from a series of runs with different numbers of processors, seems valid except for the cost of context swapping and process suspension, which depends upon the amount of data base interference and the number of processors.

sufficiently many completely non-interfering processes (i.e., processes which do not interfere in any way with the execution progress of one another), one would expect the achieved parallelism (speed-up) of that set of processes executing on n identical processors to be a factor of n, as compared to the same set of processes executing on a single processor (assuming the same scheduling and multiprocessing overheads). While the HSII organization attempted to allow the various knowledge sources to be as independent as possible, the various processes were to cooperate with one another (primarily via the blackboard data base) in the effort to effect the problem solution.[1] This necessary cooperation (and the various forms of execution interference resulting from it) was expected to result in the achieved parallelism in a multiprocessor environment being somewhat less than the potential parallelism without interference.

Several experiments were run to measure the parallelism achieved in this particular implementation of the HSII problem-solving organization using varying numbers of identical processors. Each of these experiments was run with the knowledge-source set described previously, using the same input data (introduced into the data base so as to simulate real-time speech input), the same blackboard locking structure, and the same scheduling algorithm, while varying the number of (identical) processors. An example of the graphical output produced by the simulation, for the case of eight processors, is displayed in Figure 3. To comment on these activity plots, the "# runnable processes" plot gives the number of processes either running or ready to run at each simulation scheduling point; the "# running processes" plot gives the number of actively executing processes at each scheduling point; the "# ready processes" plot shows the number of processes awaiting assignment to a processor at each scheduling point; and the "# suspended processes" plot gives the number of processes blocked from executing because of data access interference or because they are waiting on the receipt of a tagging message.

Referring to Figure 3c, notice the spiked nature of the ready-processes plot. This is a result of delaying the execution of a precondition (due to the limited processing power available) beyond the point in time at which its pre-precondition is

[1] Note that the size of the HSII blackboard is expected to grow to only several thousand nodes (hypotheses and links), at, say, 25 field entries apiece, depending, of course, on the task domain. Thus, it is assumed (for the purposes of the current investigations, at least) that the blackboard is entirely resident in primary memory; thus, input/output operations are not an issue here, the system being essentially compute-bound.

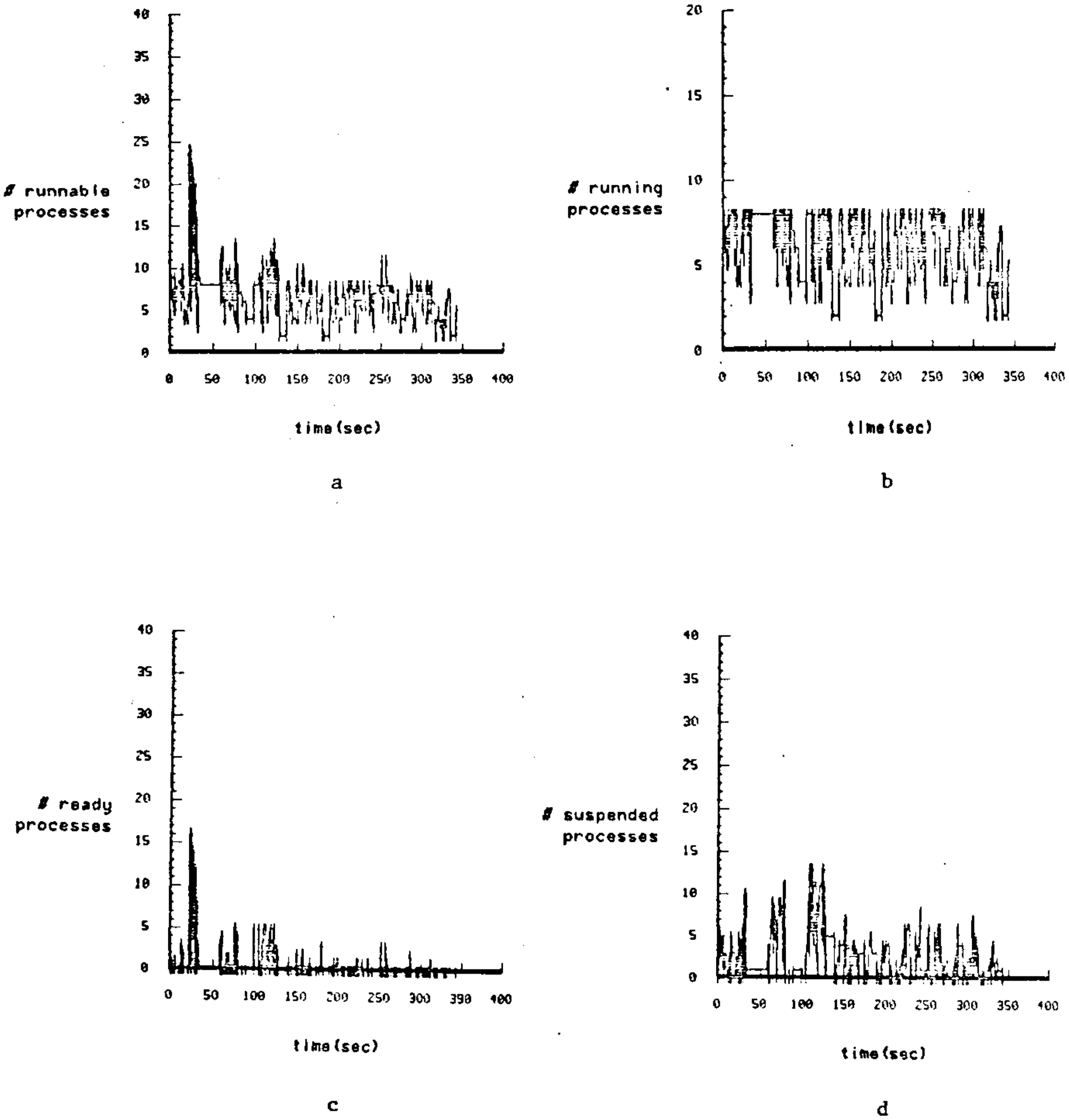

Figure 3a-d. 8 Processors

first satisfied: the longer a precondition is delayed, the more data events it is likely to accumulate in the meantime, and the more knowledge-source processes it is likely to instantiate once it does get executed; hence the spiked nature of the resultant ready-processes plots for configurations of few processors. As parallel processing power increases, preconditions can more often be run as soon as their pre-preconditions are initially satisfied, and the spiking phenomenon subsides.

As an example of how these activity plots have been used in upgrading the performance of the implementation, compare Figure 4 to Figure 3c. Figure 4 depicts the process activity under the control of a scheduler which did not attempt to perform load balancing with respect to ready preconditions; and as a result of not increasing the relative scheduling priority of preconditions as they received more and more data events, the activity spike phenomenon referred to above became predominant, to the extent of reducing process activity to a synchronous system while the long-time waiting precondition instantiates a great many knowledge-source processes all at once.[1] Figure 3c shows the activity on the same number of processors, but using a somewhat more intelligent scheduling algorithm, with a resulting reduction in the observed spiking phenomena. This improved scheduling strategy is the one used for all plots presented herein.

In addition to the plots described above, various other measures were made to allow an explicit determination of processor utilization and effective parallelism for varying numbers of processors. Referring to Table 2, one can get a feeling for the activity generated by employing increasing numbers of processors. All simulations represented in Table 2 were run for equivalent amounts of processing effort with respect to the results created in the blackboard data base by the knowledge source activity. The final clock time of the multiprocessor configuration being simulated is given in simulated real-time seconds, and the accumulated processor idle and lost times are also given. *Idle time* is attributed to a processor when it has no process assigned to it and there are no ready processes to be run; *lost time* is attributed when the process on a processor is suspended for any reason and there are no ready processes

[1] This can be inferred from Figure 4 by noting that the sample points (vertical tick marks) are taken at each simulation scheduling point, and the lack of samples between times 220 and 380 indicates that the process that started running at 220 had no concurrently running processes competing with it until time 380, when there were suddenly 25 new processes contending for computing resources.

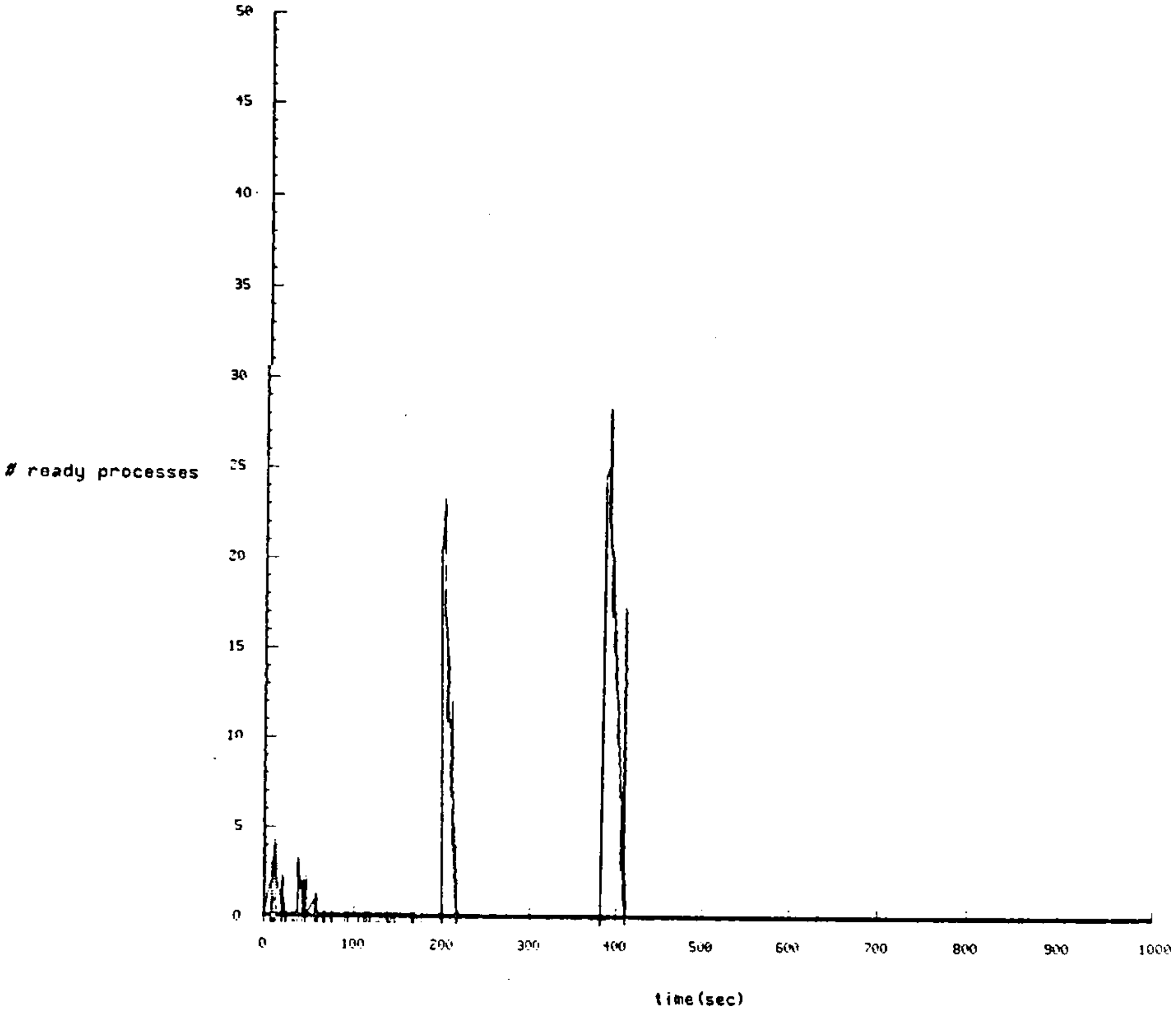

Figure 4. 8 Processors-old scheduling strategy

number of prcrs	1	2	4	8	16	32
(all times in secs)						(special*)
KS instantiations	355	401	423	421	415	434
PRE activations	82	126	173	213	200	229
multiprcr clock time	1076	634	389	350	351	43
total idle time	9	15	37	380	2608	867
total lost time	0	5	34	900	1546	0
avg cxt swaps	0	309	942	368	9	0
avg prcr utilization	99%	98%	95%	54%	26%	37%
effective # prcrs	0.99	1.96	3.80	4.32	4.16	11.84
utilization speed-up	1.00	1.98	3.84	4.36	4.20	11.96

* The 32-processor column represents an experiment which was run under special conditions, to be explained below, and should not be compared directly to the other columns of the table.

Table 2. Processor Utilization

which could be swapped in to replace the suspended process. Processor utilization (calculated using the final clock time and processor idle and lost times) is given in Table 2; Figure 5 shows the corresponding effective parallelism (speed-up), based on the processor utilization factors of Table 2.

The speed-up for this particular selection of knowledge sources is appreciable up to four processors, but drops off substantially as one approaches sixteen processors. In fact, a rather distressing feature of this effective parallelism plot is that the speed-up actually decreases slightly in going from eight processors to a sixteen-processor configuration (from a speed-up of 4.36 over the uniprocessor case, down to 4.20). This may be explained by noting that both the eight- and sixteen-processor runs had approximately equal final clock times; but in the sixteen-processor

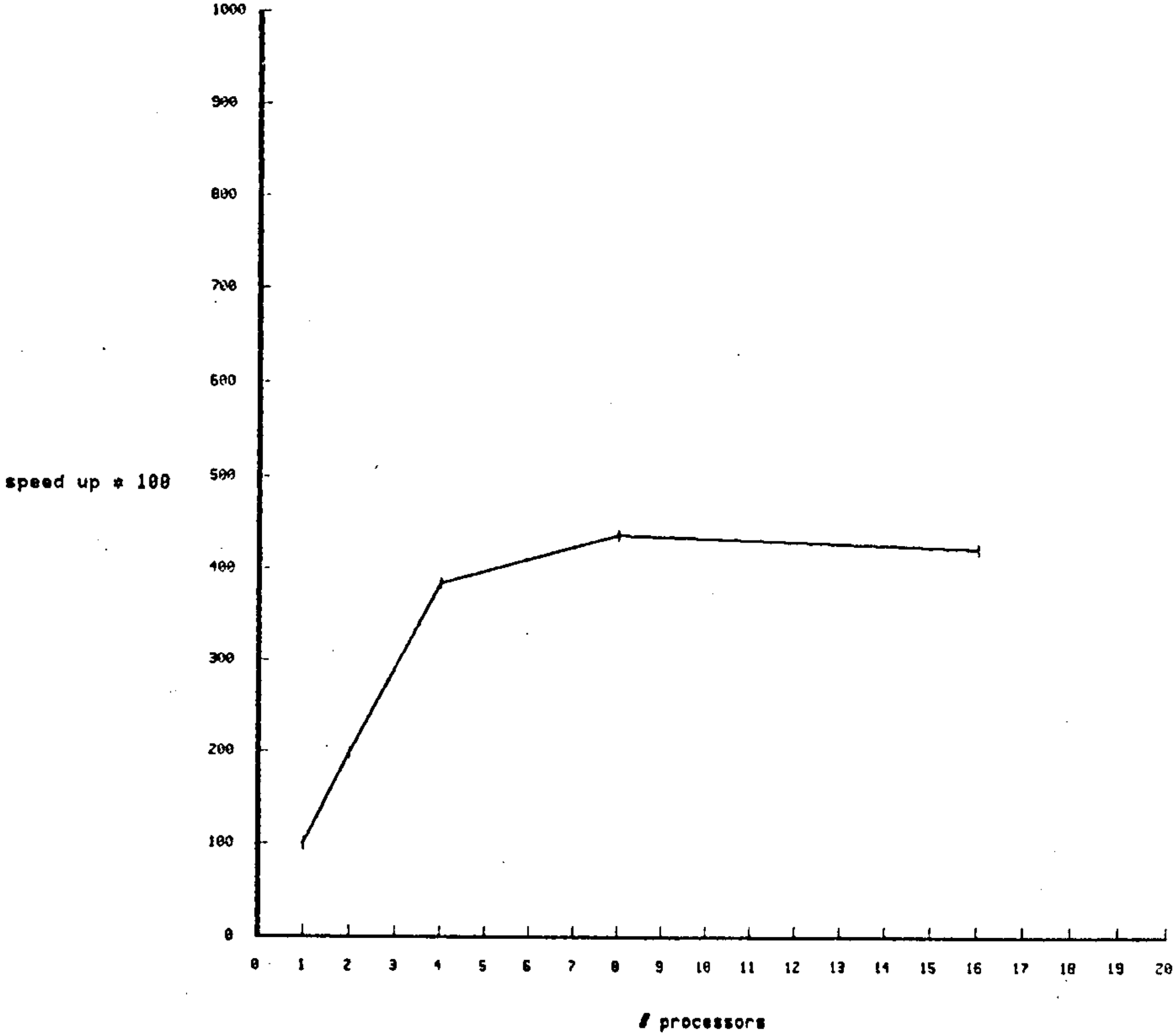

Figure 5. Effective Parallelism According to Processor Utilization

case, the number of runnable processes never exceeded sixteen processes, so any ready process could always be accommodated immediately. As a result, the number of knowledge-source instantiations and precondition activations fell off a bit from the eight-processor case, because the preconditions were more likely to be fully satisfied the first time they were activated (since all ready-processes, knowledge-source processes in particular, could be executed immediately and complete their intended actions sooner, so that when a precondition came to be activated, it would more likely find its full data pattern to be satisfied); thus, preconditions would not often be aborted, having to be re-tested upon receiving a subsequent data event. However, running fewer preconditions resulted in much more idle time for the sixteen-processor configuration (the increase in lost time indicated in Table 2 is an artifact of having too many processors available, since suspended processes would tend to remain on otherwise idle processors rather than being swapped off the processor -- note the rather dramatic decrease in context swaps indicated by Table 2 for the sixteen-processor case). The result is a lower proportionate utilization of the processor configuration, and hence a decrease in the effective parallelism from the eight-processor configuration to the sixteen-processor configuration.

Due to the limited state of development of the total set of knowledge sources, the set of knowledge sources used in the simulation was necessarily limited; so the fact that these plots indicate that not more than about four to eight processors are being effectively utilized is not to say that the full HSII speech-understanding system needs only eight processors. One might ask that if only 4.16 processors of the sixteen-processor configuration are being totally utilized (see Table 2), what is the maximum potential effective parallelism, given this set of knowledge sources? To answer this question, an experiment was performed in which effectively infinite processing power was provided to this knowledge-source set and all data access interference was eliminated (by removing the locking structure overheads and blocking actions); the scheduling algorithm was kept unchanged, as was the input data, although the input data stream was entered so as to be instantaneously available in its entirety (rather than being introduced in a simulated real-time, "left-to-right" manner). The results of this experiment are summarized by the 32-processor column of Table 2 (32 processors was an effective infinite computing resource in this case, since eight of the processors were never used during the simulation). Notice that no lost time was attributed to the run, due to the lack of locking interference; and the resultant processor utilization was 37%

of 32 processors, or 11.84 totally utilized processors. Thus, data base interference caused by particular data base accessing patterns and associated locking structures of the knowledge source set used in the experiment significantly affected processor utilization; if the use of the locking structures could be accomplished in a more non-interfering manner, the speed-up indicated by the eight- or sixteen-processor configurations could be increased substantially. The next section will analyze in detail the exact causes for this data base inteference, and propose changes to the knowledge-source locking structure so as to reduce potential inteference.

Table 3 presents some other system configurations to show effective processor utilizations under varying conditions. The first row repeats the statistics of the sixteen-processor case of Table 2; the second row is a summary of the 32-processor case of Table 2, as described above. Three further data points are offered to indicate the effects of increasing the size of the knowledge-source set. The last three rows of Table 3 involve experiments using an expanded knowledge-source set consisting of the knowledge sources of all the previous runs plus the *Syntactic Word Hypothesizer* (see Appendix A) and its precondition. Using this expanded knowledge-source set, simulations were performed to evaluate the effects of this knowledge-source set on a sixteen-processor configuration with the locking structure in effect, presenting the input data in the usual "left-to-right" manner, as well as in the instantaneous manner used in the infinite-processor test. Comparing the results (in Table 3) to the original sixteen-processor run, the "left-to-right" input scheme achieved a processor utilization of 33%, up 7% from the smaller knowledge-source set case; and by presenting all input data simultaneously, the utilization rose to 35%. The fifth row of Table 3 represents the results of providing effectively infinite computing power (only 25 processors were ever used during the run) to the expanded knowledge-source set and eliminating all data access interference, in the same manner as for the experiment of the second row. In this "optimal" situation for the expanded knowledge-source set, processor utilization was measured at 46%, or 14.72 totally utilized processors. Again, it may be noted that a more effective (less interfering) use of the locking structures can result in substantial increases in processor utilization and effective parallelism.

The addition of the Syntactic Word Hypothesizer was able to achieve the increases in utilization noted in Table 3 because it operates on lexicons that are used by only one other knowledge source (the Phoneme Hypothesizer) in the basic knowledge-source set; hence, the process interference introduced by adding this

experiment description	multiprcr clock	total idle	total lost	% util	effective # prcrs
8 KS's, 6 PRE's 16 prcrs, w/ lock l-to-r input	351	2608	1546	26%	4.16
8 KS's, 6PRE's 32 prcrs, w/o lock instantaneous input	43	867	0	37%	11.84
9 KS's, 7 PRE's 16 prcrs, w/ lock l-to-r input	148	854	726	33%	5.28
9 KS's, 7 PRE's 16 prcrs, w/ lock instantaneous input	155	839	784	35%	5.60
9 KS's, 7 PRE's 32 prcrs, w/o lock instantaneous input	13	226	0	46%	14.72

Table 3. System Configuration Variations

knowledge source was minimal. Unfortunately, the development of knowledge sources at lexicon levels which more directly conflict with those of existing knowledge sources has been limited, so direct experimentation on the interfering effects of such knowledge sources could not be performed; but based on the observations comparing the 32-processor without-lock experiments to the original sixteen-processor with-lock runs, substantial interference due to ineffective use of the locking structure would be expected in such cases of adding "competing" knowledge sources. One mitigating circumstance which could alleviate such interference was noted in the "instantaneous" input case of the expanded knowledge-source set case, as compared to the "left-to-right" input case: if process activity can be spread across the utterance-time dimension of the blackboard, process interference would decrease -- but interference due to data access synchronization interference can easily overwhelm this improvement. Further experiments along these lines will be attempted as the appropriate knowledge sources become available for use.

Execution Interference Measurements

In addition to the primitive operation timings and achieved parallelism measurements given above, various other measurements were made to determine the various aspects of system performance as related to multiprocessing. As has already been mentioned, a major concern in a multiprocess environment in which the various processes are not entirely independent is that of *execution interference*. Execution interference may arise whenever any given process enters a critical section within which it requires the integrity of a given data structure be maintained (thereby necessitating a means by which to disallow access to others until the critical section is exited). Execution interference may also arise whenever processes must synchronize their activities and perhaps cause themselves to wait on an event based on an action which is to be performed by some external process. Thus execution interference may arise due to causes *external* to the process being delayed (as in the case of trying to access a data structure which is currently held for exclusive access by another process), or the interference may arise due to causes *internal* to the process being delayed (as when a process delays itself by waiting for the occurrence of an externally caused event). As a result of the HSII design philosophy, which states that the various knowledge-source processes should be as independent as possible in specification and execution, most of the execution interference experienced in HSII is of the external variety, wherein a process is delayed due to external causes unknown to itself (and the delay itself is transparent to the process being delayed).

As previously described, there are two methods in the HSII system for preserving data integrity: a) guaranteeing exclusive access through the use of node- and region-locking primitives, and b) placing data assumptions in the blackboard, through tagging primitives, which when violated cause a signal to be sent to the process making the assumption. There is an interesting balance in terms of execution overhead and execution interference between these two techniques. The region-locking technique is least costly in terms of execution overhead and is the easiest to embed in a program but causes the most execution interference. This is in contrast to the use of tagging which is the most costly in terms of execution overhead and is the most difficult to embed in a program but causes the least execution interference. Both these methods were used for guaranteeing data integrity in the precondition and knowledge-source set that was used in the simulation experiments.

In structuring each knowledge source so as to preserve its data integrity, no *a priori* assumptions were made about the non-modifiability of any blackboard data that knowledge source used in its processing (i.e., it was assumed that any blackboard information that the knowledge source read could perhaps be modified by some other concurrent knowledge-source). This self-contained approach to the design of a knowledge source's locking and tagging structure is required if the modularity of the system, with respect to deletion or addition of knowledge sources, is to be preserved.

The knowledge sources that were used in the simulation experiments were not originally designed so that they could be interrupted at arbitrary points in their processing, and consequently they lacked the appropriate locking and tagging structure to guarantee data integrity in a multiprocess(or) environment. The addition, as an afterthought, of the appropriate locking and tagging structure to these knowledge sources was sometimes quite difficult. This was an especially serious problem when an attempt was made to put tagging primitives into knowledge sources which had internal backtracking control structures for searching the node graph structure in the blackboard. This difficulty arises because previously made data assumptions (tags in the blackboard) associated with a partial path (sequence of nodes in the blackboard) must be removed upon discovering that the path cannot be successfully completed. Thus, most of the knowledge sources in the experiment did not use tagging as a method of guaranteeing integrity, but rather used a combination of node- and region-locking. However, preconditions, which have a much simpler structure and generally do not write in the blackboard, were modified to use the tagging mechanism. In addition, to further simplify knowledge-source locking structures, region-locking was used wherever possible. This excessive use of region-locking was mainly responsible for the significant amount of interference among processes which caused the effective processor utilization to go from an optimal 12 to a realized 4 (see Table 2).

Figure 6 shows an interesting case demonstrating that the indiscriminate use of region-locking can obstruct the execution progress of many processes and thereby temporarily reduce the effective parallelism of the system. It represents a snapshot of the blackboard locking structure taken during the execution of the simulation. The grid structure represents the two-dimensional abstract data structure, the dimensions being lexicon level and region element number (corresponding to the utterance-time dimension). At the point of each snapshot, the outstanding node and region locks are indicated, as well as the areas requested (but not yet obtained) by suspended processes. The various (non-interfering) tags placed throughout the data base are also indicated. The key indicates the sets of active and suspended processes (the names referring to the precondition and knowledge source names, and the numbers in the names indicating a process instantiation index unique to that particular process). This particular snapshot was taken from the sixteen-processor simulation run with the smaller knowledge-source set. Notice that PSYN263 has locked regions at the PHON, MXN, and PSEG lexicon levels for its exclusive access; the nodes locked by PSYN263 (hypotheses being indicated by H<sequence number>, and links by L<sequence number>) within these regions are those being created by PSYN263, hence the reason for the region locks. Unfortunately, this locking action resulted in the suspension of six other processes awaiting access to parts of the PHON and PSEG lexicon levels which overlap PSYN263's region-locks. Each of these suspended processes is waiting to acquire access-rights to a node in these locked regions; in fact, PRE!PSYN!PSYN and CSEG259 are both waiting on the same node (H141). The diagram also shows the various (non-interfering) tags which were placed on the various nodes at the PHON and PSEG lexicon levels by three of the processes at some previous time. Figure 7, which is another snapshot of locking structure, shows a case where execution interference was not so significant.

The reason the locking structure plots are localized in the lower left-hand corner of the blackboard structure is that the construction of the data base in the speech-processing task is initially left-to-right due to the time-sequential nature of the speech input. Also, the particular set of knowledge sources chosen for use in the simulation experiments happened to be an effectively bottom-up speech recognition system (some of the top-down knowledge sources having not yet been developed to a stable enough state to have been used in the simulations); hence, activity starts in the lower left-hand corner of the blackboard. Further simulations are planned which will work in a combined top-down and bottom-up fashion, thereby increasing the potential

	1				5					10					15					20					25					30
SHDSENT																														
SHDWORD																														
WORD																														
WRDSURN																														
SURN																														
PHON					c	A b c	A d 4	A A d 4	A A	A																				
MXN						A	A	A	A	A																				
PSEG					e f	A g e f 7	A g e f 7	A A 1 7	A A 1 7	A 7																				
SEG																														

Region Locks	Node Locks	Region Waits	Node Waits	Tags
A	A: H150,H151,H152 L101,L102		b: H146 c: H142 d: L100 e: H141 f: H141 g: H69	1: H72,H75,H75,H77 4: H149 7: H69,H70,H70, H72,H141

Running Processes

A/1: !PSYN263

Suspended Processes

B/2:	PRE!PSC!PSC
C/3:	PRE!UTTBOUNDARIES!PSC
D/4:	PRE!RPOL!RPOL
E/5:	PRE!PSYN!PSYN
F/6:	!CSEG259
G/7:	!PSYN262

Figure 6. 16 Processors- blackboard lock map at time 155.1

	1				5					10					15					20					25					30
SHDSENT																														
SHDWORD																														
WORD																														
WRDSURN		4																												
SURN	4	4																												
PHON				F	F i	E i	E j	E	E																					
MXN																														
PSEG				F G	F G 3	E 3	E 3	E 3	E																					
SEG																														

Running Processes

A/1:	PRE!PSC!PSC
B/2:	PRE!RPOL!RPOL
C/3:	PRE!PSYN!PSYN
D/4:	!SEARCH279
E/5:	!TIME280
F/6:	!UV282
G/7:	!UV283
H/8:	!UV284

Suspended Processes

I/9:	PRE!UTTBOUNDARIES!PSC
J/10:	!TIME281

Region Locks

E

Node Locks

F:	H123,H155,L106
G:	L107

Region Waits

Node Waits

i:	H143
j:	H154

Tags

3:	H88,H92,H92,H92, H94,H94,H94,H94, H98,H98,H98,H98, H106,H112
4:	H27,H27,H38

Figure 7. 16 Processors- blackboard lock map at time 183.1

parallelism (since the top-down knowledge sources will presumably not interfere with the execution of the bottom-up knowledge sources as much as additional competing bottom-up knowledge sources would). The expanded knowledge-source set experiments presented above were a first step in introducing such top-down knowledge; as more knowledge sources become available, their various interference effects will be investigated. Also, other tasks which could use the HSII organization might not necessarily have the left-to-right input characteristics of speech, so future simulations will also test a more distributed input pattern, thereby also increasing the potential parallelism by spreading the process activity across the breadth of the blackboard; the several experiments presented above which introduced the input in an "instantaneous" manner were the initial attempts in this direction.

A more analytic approach to analyzing the data access interference experienced by precondition and knowledge source processes, for varying numbers of processors, is given in Table 4.

number of prcrs (all times in secs)	1	2	4	8	16
avg BB accesses/KS	54.4	52.8	54.5	53.9	56.4
avg BB accesses/PRE	96.7	68.7	55.7	48.2	51.1
avg prim locks/KS	27.9	27.4	28.0	25.7	26.9
avg prim locks/PRE	96.7	68.7	55.7	48.2	51.1
avg dsched/prim lock(KS)	0	0.020	0.060	0.055	0.053
avg dsched/prim lock(PRE)	0	0.009	0.026	0.045	0.040
avg dsched duration/KS	0	5.08	5.69	1.75	1.90
avg dsched duration/PRE	0	3.95	1.91	1.35	1.86
avg cxt swaps	0	309	942	368	9
avg cxt swaps/dsched	0	1.03	0.97	0.36	0.01

Table 4. Data Access Characteristics

Essentially, Table 4 is an extension of Table 2, which was discussed in the previous section (i.e., the underlying simulation runs were the same for both tables). Execution interference was measured by recording the amount of process suspension (also called *descheduling*), which results from processes being temporarily blocked in their attempts to gain access to some part of the blackboard data base.[1] As might be expected, as process activity increases with increasing numbers of processors, the possibility of execution interference increases (see table entries on "deschedules/primitive lock"). This phenomenon stops at eight processors because in these simulation experiments there were rarely more than eight processes executing at any given moment. At the same time, with more and more processing power available, the likelihood of suspended processes being unblocked and becoming available for further processing increases as the number of processors increases (see table entries on "deschedule duration"). This phenomenon is also indicated by the significant decrease in processor context swaps per deschedule (i.e., with more processors, it becomes less likely that when a process is suspended there will be another process ready to execute).

The major point that can be drawn from this table is that the decrease in processor utilization caused by the locking structure is not due to the high rate of data access interference (i.e., at most only 6% of the primitive locks result in deschedules) but rather from the long duration over which descheduled processes are blocked. This deschedule duration, in the optimal case of 16 processors, where processes do not have to wait for for an available processor, is approximately 2 seconds, which is very close

[1] The number of deschedules attributed to a process is related to the inner workings of the locking mechanism. Not only is the granularity of the locking structure important (i.e., how small a piece of the blackboard data base can be requested for access allocation), but the granularity of the process blocking mechanism is important. For example, processes could be blocked upon trying to gain access to a node and then relegated to waiting in a set of processes which are waiting on <u>any</u> node at the level of the requested node; or the wait set could be divided according to the individual nodes being waited upon. If, in an attempt to conserve semaphore structures, the former strategy is chosen, it could become quite expensive to determine whether, upon receiving an unlock wake-up signal for the wait set, a particular member of the wait set is really re-schedulable as a result of that wake-up signal; hence, it may be cheaper to release all waiting processes in the set, even though all but one will just become descheduled again. If the single-node wait set is used, the costs of maintaining separate semaphores for every possible data object may become prohibitively expensive, although process re-scheduling would not be done unnecessarily in such a scheme.

to the average run time of a knowledge source. This long duration occurs because the knowledge-source locking structures involve executing region locks at the beginning of the knowledge source execution. These region-locks define the entire blackboard area (and perhaps even more) that the knowledge source will either examine or modify during its entire execution.[1] These locks are then released only at the termination of the knowledge source execution. Thus, if data access interference (i.e., a primitive lock deschedule) occurred because of a previously executed region-lock, then the suspended process would very likely not be unblocked until the knowledge source executing the region-lock had completed its processing.

Finally, it is once again admitted that the results presented here are derived from a rather limited selection of knowledge-source processes, the coding style of which may be affected by the various efficiencies and inefficiencies of the particular implementation of the HSII system organization. In particular, since the HSII speech-understanding system is under constant development, various code sections involving the system operations have been subject to extensive optimization attempts, while other sections have not yet had the benefit of such optimization. Additionally, the results are biased by the task domain (viz., speech understanding) and the data structure chosen to represent the dynamic solution state of the task. However, it is hoped that the system organization (including the data base design) is of sufficiently general character that these particular results at least give a feeling for the results that might be expected using a different set of knowledge-source processes to solve the same or different problems.

SUMMARY AND CONCLUSIONS

This paper has presented a design for the organization of knowledge-based AI problem-solving strategies which is felt to be particularly applicable for implementation on closely-coupled multiprocessor computer systems. The method of design is a result of formulating the problem-solving organization in terms of the

[1] Note that the number of primitive lock operations for preconditions is equal to the number of blackboard accesses (from the precondition process averages of Table 4): preconditions do not usually need a long-lasting locked environment (since they do not modify the blackboard except to place tags into it), thus each access is individually protected by the HSII operating system (via temporary-locking), rather than having the precondition perform an explicit *LOCK!* operation before each access.

hypothesize-and-test paradigm for heuristic search, where the various hypothesizers and testers are represented as knowledge sources applicable to the task domain of the problem being solved. A *knowledge source* may be described as an agent that embodies the knowledge of a particular aspect of the problem domain and is useful in solving a problem from that domain by performing actions based on its knowledge so as to further the progress of the overall problem solution. The hypothesize-and-test paradigm provides the conceptual means of coordinating these various knowledge source activities by suggesting that it is the function of some knowledge sources to create *hypotheses* representing a possible (perhaps partial) solution state for the given problem. Hypotheses are created in a global data base and are available for inspection by all knowledge sources. It is the responsibility of other knowledge sources to evaluate these hypotheses in light of their own knowledge of the task domain, and either accept or reject the hypotheses, or propose their own alternative hypotheses (by either modifying the existing hypotheses or creating entirely new ones).

The Hearsay II speech-understanding system (HSII), which has been developed at Carnegie-Mellon University using the techniques for system organization described here, has provided a context for evaluating this system architecture. The HSII organization provides the facilities necessary for knowledge-source cooperation through the hypothesize-and-test paradigm to be carried out in a highly asynchronous and *data-directed* manner, where knowledge sources are specified as independent processing entities capable of parallel execution; the activities of any given collection of such knowledge sources are coordinated by the hypothesize-and-test paradigm through the use of a shared global data base called the *blackboard*.

In specifying the blackboard as the primary means of interprocess communication, particular attention has been paid to resolving the *data access synchronization* problems and *data integrity* issues arising from the asynchronous data access patterns possible from the various independently executing parallel knowledge-source processes. A non-preemptive data access allocation scheme was devised in which the units of allocation could be linearly ordered and hence allocated according to that ordering so as to avoid data deadlocks. The particular units of data allocation (locking) were chosen as being either blackboard nodes (*node-locking*) or abstract regions in the blackboard (*region-locking*). Blackboard nodes also represent the units of data creation within the blackboard. The region-locking mechanism views the potential blackboard as an abstract data space in which access rights to abstract regions could be granted without regard to the actual data content of these regions.

Another area of concern relating to the use of a shared blackboard-like data facility relates to the assumptions made by the various executing knowledge sources concerning issues of data integrity and localized data contexts. Since the blackboard is intended to represent only the most current global status of the problem solution state, mechanisms were introduced to allow individual knowledge sources to retain recent histories of modifications made to the dynamic blackboard structure in the form of *local contexts*. Knowledge sources are also permitted to mark (*tag*) arbitrary fields (or nodes or regions) of the blackboard itself (without requiring continuing access rights to the field being tagged) and thereby monitor (in a non-interfering way) those locations for subsequent changes; the knowledge source will then be sent *messages* should any modifications be performed upon a tagged field. Local contexts provide knowledge sources with the ability to create a local data state which reflects the net effects of data events which have occurred in the data base since the time of the knowledge source's activation. Combined with the blackboard data tagging capabilities, local contexts also provide a means by which knowledge sources can execute quite independently of any other concurrently executing knowledge sources (and without interfering with the execution progress of any of these processes).

In an attempt to improve the problem-solving efficiency of a multiprocessor implementation of the system by increasing the amount of potential parallelism from knowledge source activity, the logical functions of precondition evaluation and knowledge source execution are split into separate processing entities (called, of course, *precondition* and *knowledge-source processes*). A *precondition process* is responsible for monitoring and accumulating blackboard data events which might be of interest to the knowledge source associated with the precondition; and when the appropriate data conditions for the activation of the knowledge source exist in the blackboard, the precondition will instantiate a *knowledge-source process* based on its associated knowledge source, giving to the new process the data context in which the precondition was satisfied.

The process activity of HSII is intended to be very *data-directed* in nature, basing the decisions as to whether a knowledge source action can be performed on the dynamic data state represented in the blackboard data base. It is the responsibility of a precondition to test this data state for conditions which would warrant the instantiation of the knowledge source associated with the precondition. The activation of the precondition itself is also data-directed, being based on *monitoring* for the more primitive blackboard modification operations which knowledge-source processes may invoke to effect the results of their computation. This blackboard monitoring is implemented by having the various blackboard modification operators be responsible for the activation of preconditions which are monitoring for data events being caused by the modification operation.

In order to indicate the nature of the performance of the HSII organization when run in a closely-coupled multiprocessor environment, a simulation system was embedded into the multiprocess implementation of HSII on the DECsystem-10. While the results of the simulation are admittedly based on a small (but computationally expensive) set of sample points, they have generally indicated the applicability of this system organization to such a hardware architecture. Given the knowledge-based decomposition of a problem-solving organization as prescribed by the HSII structure, effective parallelism factors of four to six were realized even with a relatively small set of precondition and knowledge-source processes, with indications that up to twelve processors could be totally utilized, given appropriate usage (or structuring) of the data access synchronization mechanisms. Experiments thus far have indicated that careful use of the locking structure is required in order to approach the optimal utilization of any given processor configuration (unless there exist so many ready processes that the number of suspended processes does not matter much, as is the case in configurations of four or fewer processors). An extended use of non-interfering tagging seems to be indicated, along with a reduction in the use of region-locking (perhaps substituting region-examining or node-locking wherever possible). Measurements were also made of various system level primitive operations which are required in order to implement the data-directed multiprocess structure of HSII. While all these results are of a preliminary nature (and hence are subject to variation as various components of the given implementation are improved in their relative efficiencies), they seem to indicate that the HSII organization is indeed applicable for efficient use in a closely-coupled multiprocessor environment.

ACKNOWLEDGMENTS

We wish to acknowledge the contributions of the following people: Lee Erman for his major role in the design and development of HSII, Raj Reddy for many of the basic ideas which have led to the organization described here, and Gregory Gill for his untiring efforts in systems implementation.

Appendix A:
HSII BLACKBOARD AND KS DECOMPOSITION

Conceptual ____________
Phrasal ____________
Lexical ____________
Syllabic ____________
Surface-phonemic ____________
Phonetic ____________
Segmental ____________
Parametric ____________

Figure 1. The Levels in Hearsay II.

Figure 1 shows a schematic of the information levels of Hearsay II.

Parametric Level - The parametric level holds the most basic representation of the utterance that the system has; it is the only direct input to the machine about the acoustic signal. Several different sets of parameters are being used in Hearsay II interchangeably: 1/3-octave filter-band energies measured every 10 msec., LPC-derived vocal-tract parameters, and wide-band energies and zero-crossing counts.

Segmental Level - This level represents the utterance as labeled acoustic segments. Although the set of labels may be phonetic-like, the level is not intended to be phonetic -- the segmentation and labeling reflect acoustic manifestations and do not, for example, attempt to compensate for the context of the segments or attempt to combine acoustically dissimilar segments into (phonetic) units. As with all levels, any particular portion of the utterance may be represented by more than one competing hypothesis (i.e., multiple segmentations and labelings may coexist).

Phonetic Level - At this level, the utterance is represented by a phonetic description. This is a *broad* phonetic description in that the size (duration) of the units is on the order of the "size" of phonemes; it is a *fine* phonetic description to the extent that each element is labeled with a fairly detailed allophonic classification (e.g., "stressed, nasalized [I]").

Surface-Phonemic Level - This level, named by seemingly contradicting terms, represents the utterance by phoneme-like units, with the addition of modifiers such as stress and boundary (word, morpheme, syllable) markings.

Syllabic Level - The unit of representation here is the syllable.

Lexical Level - The unit of information at this level is the word.

Phrasal Level - Syntactic elements appear at this level. In fact, since a level may contain arbitrarily many "sub-levels" of elements using the AND and OR links, traditional kinds of syntactic trees can be directly represented here.

Conceptual Level - The units at this level are "concepts." As with the phrasal level, it may be appropriate to use the graph structure of the data base to indicate relationships among different concepts.

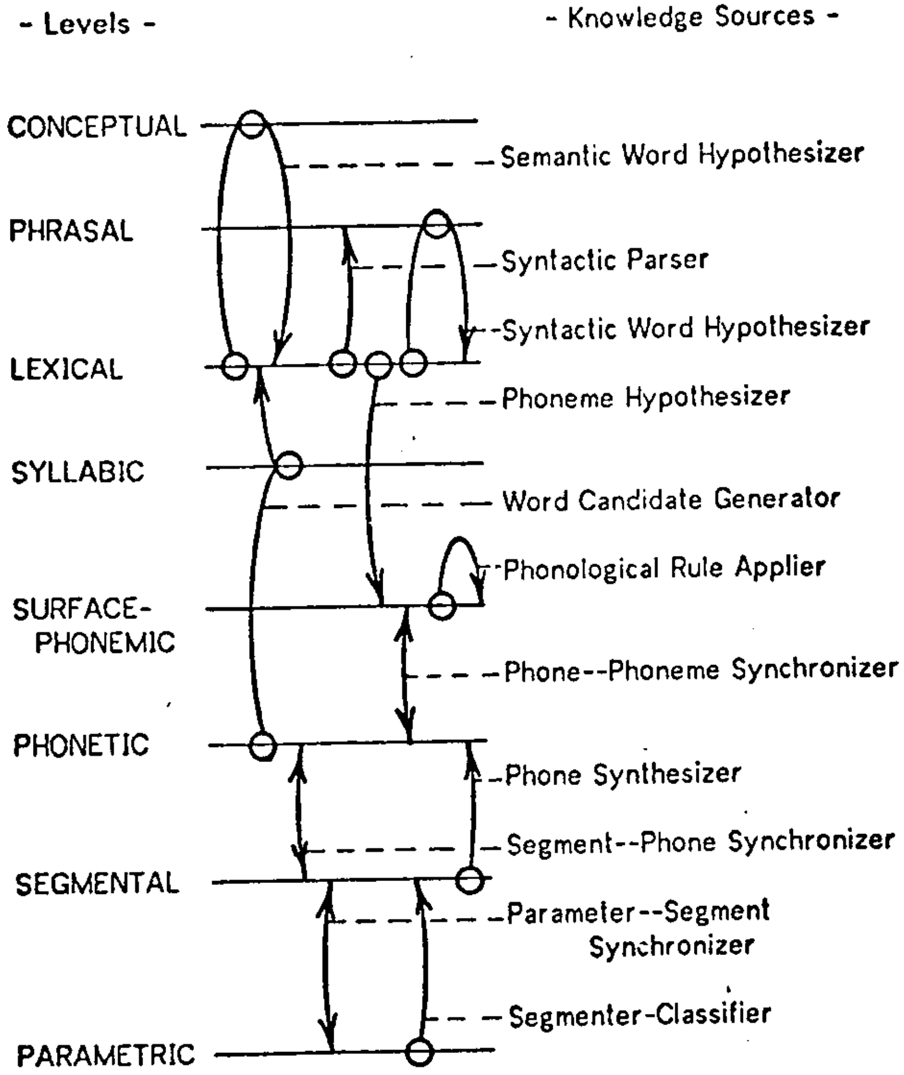

Figure 2. A Set of Knowledge Sources for HearsayII.

As examples of knowledge sources, Figure 2 shows the first set implemented for Hearsay II. The levels are indicated as horizontal lines in the figure and are labeled at the left. The knowledge sources are indicated by arcs connecting levels; the starting point(s) of an arc indicates the level(s) of major "input" for the knowledge source, and the end point indicates the "output" level where the knowledge source's major actions occur. In general, the action of most of these particular knowledge sources is to create links between hypotheses on its input level(s) and: a) existing hypotheses on its output level, if appropriate ones are already there, or b) hypotheses that it creates on its output level.

The *Segmenter-Classifier* knowledge source uses the description of the speech signal to produce a labeled acoustic segmentation. For any portion of the utterance, several possible alternative segmentations and labels may be produced.

The *Phone Synthesizer* uses labeled acoustic segments to generate elements at the phonetic level. This procedure is sometimes a fairly direct renaming of an hypothesis at the segmental level, perhaps using the context of adjacent segments. In other cases, phone synthesis requires the combining of several segments (e.g., the generation of [t] from a segment of silence followed by a segment of aspiration) or the insertion of phones not indicated directly by the segmentation (e.g., hypothesizing the existence of an [l] if a vowel seems velarized and there is no [l] in the neighborhood). This knowledge source is triggered whenever a new hypothesis is created at the segmental level.

The *Word Candidate Generator* uses phonetic information (primarily just at stressed locations and other areas of high phonetic reliability) to generate word hypotheses. This is accomplished in a two-stage process, with a stop at the syllabic level, from which lexical retrieval is more effective.

The *Semantic Word Hypothesizer* uses semantic and pragmatic information about the task (e.g., news retrieval or chess) to predict words at the lexical level.

The *Syntactic Word Hypothesizer* uses knowledge at the phrasal level to predict possible new words at the lexical level which are adjacent (left or right) to words previously generated at the lexical level. This knowledge source is activated at the beginning of an utterance recognition attempt and, subsequently, whenever a new word is created at the lexical level.

The *Phoneme Hypothesizer* knowledge source is activated whenever a word hypothesis is created (at the lexical level) which is not yet supported by hypotheses at the surface-phonemic level. Its action is to create one or more sequences at the surface-phonemic level which represent alternative pronunciations of the word. (These pronunciations are currently pre-specified as entries in a dictionary.)

The *Phonological Rule Applier* rewrites sequences at the surface-phonemic level. This knowledge source is used: a) to augment the dictionary lookup of the Phoneme Hypothesizer, and b) to handle word boundary conditions that can be predicted by rule.

The *Phone-Phoneme Synchronizer* is triggered whenever an hypothesis is created at either the phonetic or the surface-phonemic level. This knowledge source attempts to link up the new hypothesis with hypotheses at the other level. This linking may be many-to-one in either direction.

The *Syntactic Parser* uses a syntactic definition of the input language to determine if a complete sentence may be assembled from words at the lexical level.

The primary duties of the *Segment-Phone Synchronizer* and the *Parameter-Segment Synchronizer* are similar: to recover from mistakes made by the (bottom-up) actions of the Phone Synthesizer and Segmenter-Classifier, respectively, by allowing feedback from the higher to the lower level.

In addition to the knowledge source modules described above, all of which embody speech knowledge, several policy modules exist. These modules, which interface to the system in a manner identical to the speech modules, execute policy decisions, e.g., propagation of ratings and calculation of processing-state attributes.

SELECTED REFERENCES

Baker, J. (1974). "The DRAGON System -- An Overview," in **Proc. IEEE Symp. Speech Recognition**, Carnegie-Mellon Univ., Pittsburgh, Pa., April 1974, pp. 22-26; also appeared in IEEE Trans. **on Acoustics, Speech, and Signal Processing**, ASSP-23, 1, pp. 24-29 (Feb. 1975).

Bell, C. G., W. Broadley, W. Wulf, A. Newell, et al. (1971). "C.mmp: The CMU Multi-mini-processor Computer," Tech. Rep., Comp. Sci. Dept., Carnegie-Mellon Univ., Pittsburgh, Pa.

Bell, C. G., R. C. Chen, S. H. Fuller, J. Grason, S. Rege, and D. P. Siewiorek (1973). "The Architecture and Application of Computer Modules: A Set of Components for Digital Systems Design," COMPCON 73, San Francisco, Calif.

Coffman, E. G., M. J. Elphick and A. Shoshani (1971). "System Deadlocks," **Computing Surveys** 3, 2, pp. 67-78.

Erman, L. D., and V. R. Lesser (1975). "A Multi-Level Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge," 4th International Joint Conference on Artificial Intelligience, Tiblesi, Russia.

Feldman, J. A., and P. D. Rovner (1969). "An Algol-based Associative Language," **Comm.** ACM 12, 8, pp. 439-449.

Feldman, J. A., et al. (1972). "Recent Developments in Sail -- An Algol-based Language for Artificial Intelligence," **Proc.** FJCC.

Fennell, R. D. (1975). "Multiprocess Software Architecture for A.I. Problem Solving," Tech. Rep. (Ph.D. Thesis), Comp. Sci. Dept., Carnegie-Mellon Univ., Pittsburgh, Pa.

Heart, F. E., S. M. Ornstein, W. R. Crowler and W. B. Barker (1973). "A New Minicomputer/Multiprocessor for the ARPA Network," **Proc. AFIPS**, NDD42, pp. 529-537.

Lesser, V. R., R. D. Fennell, L. D. Erman and D. R. Reddy (1974). "Organization of the Hearsay II Speech Understanding System," in **Proc. IEEE Symp. Speech Recognition**, Carnegie-Mellon Univ., Pittsburgh, Pa., April 1974; also appeared in IEEE Trans. **on Acoustics, Speech, and Signal Processing**, ASSP-23, 1, pp. 11-23 (Feb. 1975).

Newell, A. (1973). "Production Systems: Models of Control Structures," in W. C. Chase (ed.) **Visual Information Processing**, Academic Press, pp. 463-526.

Ohlander, R. B. (1975). "Analysis of Natural Scenes," Tech. Rep. (Ph.D. Thesis), Carnegie-Mellon Univ., Pittsburgh, Pa.

Reddy, D. R. (1973a). "Eyes and Ears for Computers," Tech. Rep., Comp. Sci. Dept., Carnegie-Mellon Univ., Pittsburgh, Pa. Keynote speech presented at Conf. on Cognitive Processes and Artificial Intelligence, Hamburg, April, 1973.

Reddy, D. R., L. D. Erman and R. B. Neely (1973b). "A Model and a System for Machine Recognition of Speech," **IEEE Trans. Audio and Electroacoust.**, AU-21, 3, pp. 229-238.

Reddy, D. R., L. D. Erman, R. D. Fennell and R. B. Neely (1973c). "The HEARSAY Speech Understanding System: An Example of the Recognition Process," **Proc. 3rd Inter. Joint Conf. on Artificial Intel.**, Stanford, Calif., pp. 185-193.

Swinehart, D. and R. Sproull (1971). **SAIL.** Stanford AI Proj. Operating Note 57.2, Stanford Univ., Stanford, Calif.