A CSR-NL INTERFACE SPECIFICATION
Version 1.5[1]

Douglas B. Paul
Lincoln Laboratory, MIT
Lexington, MA 02173

Advisory Committee:
Janet Baker, Charles Hemphill, Lynette Hirschman

ABSTRACT

Spoken Language Systems will require integration of continuous speech recognition and natural language processing. This is a proposed specification for an interface between a continuous speech recognizer (CSR) and a natural language processor (NLP) to form a spoken language system. Both components are integrated with a stack controller and contribute to the search control. The specification also defines a "Top-N" mode in which a "first part" outputs a list of top N scored sentences for postprocessing by a "second part". An additional use for this specification might be NLP evaluation testing: a common simulated CSR could be interfaced to each site's NLP to provide identical testing environments.

1 INTRODUCTION

This is a proposed specification for an interface between an acoustic matcher, such as a Hidden Markov Model (HMM) Continuous Speech Recognizer (CSR), and a grammatical component such as a natural language parser (NLP). Its purpose is to allow independently developed CSR and NLP systems to be interconnected by a well specified and well structured interface. It can also be used to provide a simulated SLS environment for developing a CSR or NLP by providing an interface to a simulator of the other component. After initial independent component development has been completed, the interface specification will guarantee that the real components can be interconnected for operation or joint development. It might also be used for NLP evaluation testing by providing a common (simulated) acoustic recognizer to use in conjunction with the NLPs under test.

The fundamental purpose of this specification is to provide an interface specification for connecting the two components so that independent sites can join their modules together. It is hoped that sites which can produce both components internally will consider this specification on its own merit and the potential value of being able to interface to modules developed at other sites.

[1]This work was sponsored by the Defense Advanced Research Projects Agency.

This specification provides for two modes of operation: integrated and decoupled. In the integrated mode, both the CSR and the NLP contribute to the search control. If (or when) the CSR and NLP technologies are sufficiently mature, this will probably be the preferred mode. The decoupled mode allows the CSR component to output a list of possible sentences with acoustic match likelihoods. The NLP can then process this list as it sees fit. Since information flow in the decoupled mode is strictly feed-forward, no NL information is available to help constrain the search in the CSR component.

The specification contains overall control architecture and interface definitions. The resulting system consists of a combined stack-controller/CSR (SC-CSR) and NLP interconnected by UNIX pipes. Simulators for each component will be provided to allow sites which are developing only one of the components to work within the context of a full SLS system and to allow sites which are developing both components to perform independent development of both modules if they so wish.

The basic algorithmic constraints required by this interface are fairly mild: the interprocess interface uses UNIX pipes, and both the CSR and NLP components operate left-to-right on their respective input data. (However, the decoupled mode allows the NLP to use non-left-to-right strategies such as island-driven. The decoupled mode may increase the CPU requirements of the overall system.)

The original idea and the definition of this interface is the work of D. Paul. An Advisory Committee of both NL and CSR people has reviewed the proposal from both viewpoints. The committee members are:

Janet Baker	Dragon Systems
Charles Hemphill	TI
Lynette Hirschman	UNISYS

The comments of these committee members have been very useful to the author. However, their membership does not imply agreement with all provisions of this specification. A draft has been distributed to all sites in the DARPA SLS program for comment before its presentation at the October 1989 meeting.

1.1 The Basic System Concept

The basic concept requires three parts:

1. A stack controller (similar to the IBM stack decoder). The "stack" is a sorted list of partial theories.
2. A CSR capable of evaluating the probability of the acoustic data for a given left sentence fragment.
3. An NLP capable of evaluating the probability of a given left sentence fragment.

The basic system operation is:

1. The stack controller starts with a null theory.
2. Take the most probable partial theory (left sentence fragment) off the stack.

3. If this theory consumes all acoustic data and is a full sentence, this is the recognized sentence. Terminate. (If more than one hypothesized sentence is desired, continue until a sufficient number of sentences are output. This is Top-N mode, see Sec. 2.5.)

4. For each possible succeeding word, add the word to the theory, ask the CSR for the acoustic probability, ask the NLP for the grammatical probability, and insert the new theory into the stack at a position determined by a combination of the probabilities. ("Fast matches" can be used to limit the number of succeeding words in order to reduce the search space.) Note: In general, the CSR probabilities are distributions over time.

5. Repeat from 2.

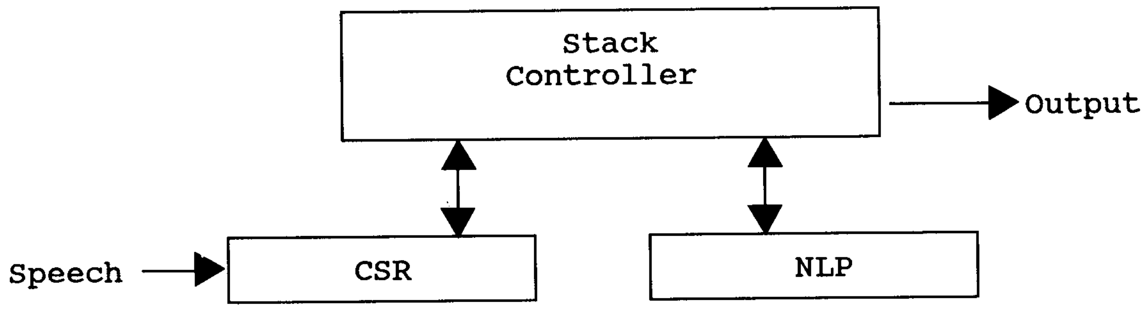

Figure 1: The conceptual system.

The above is an implementation of a "uniform" [2] search, which will find the correct (most probable) answer far to slowly to be practical. A more efficient version is outlined below.

2 DETAILED CONCEPTS

2.1 A Better Likelihood Function

The uniform search is inefficient because it delays extension of the longer theories while it extends the shorter (poorer) theories. Instead, an approximation to an "A*" search [1,2] will be used. This uses a likelihood function which gives much better comparisons between theories of varying lengths and results in a much more efficient search. If properly implemented, it is an admissible search (i.e., it is guaranteed to find the best path.) In practice, it may not be possible to compute some of the parameters so that the required approximations may compromise the guarantee. (In fact, intentionally using incorrect parameters can further reduce the search space and one may trade off computation for search error risk—see below.)

One way of implementing the A* search is to use the difference between the actual log-probability of reaching a subgoal and an upper bound upon that log-probability as the search control function. A reasonably good upper bound may be computed for the CSR component, N-gram languages and, hopefully, also for NL grammars. (In practice, estimates for the upper bound might have to be used.) This likelihood function can be evaluated in a strictly left-to-right fashion and thus the search may begin before the end of the acoustic data is found.

Thus, the basic costs used here will be log likelihoods (i.e., the difference between the upper bound log probability and the actual log probability). (The term cost as used here is more like value: high is good and low is bad.) The stack likelihood function should also include some extra control parameters:

stack_likelihood = CSR_likelihood + α*length + β*NLP_likelihood + gamma*nr_words
where α is an acoustic length penalty
length is the amount of acoustic data covered by the theory
β is a grammar weight
γ is a word insertion penalty
and nr_words is the number of words in the theory.

α controls the width of the search: $\alpha > 0$ will encourage the longer theories and thus reduce the search and $\alpha < 0$ will penalize the longer theories and thus increase the search. Since length of the entire acoustic input is is a constant across all theories, α cannot alter the relative likelihood of a complete theory—but it can prevent the best theory from being found first if it is too large. (This is, in effect, a pruning error.) β controls the relative weights of the acoustic and grammatical evidence. γ controls the relative number of insertion and deletion errors. In a perfect A* search, both CSR_likelihood and NLP_likelihood would be less than or equal to zero.

By manipulating these parameters and the likelihoods returned by the CSR and NLP, it is possible to implement a wide variety of search strategies including uniform and A*. This interface is capable of operating with any of this range of strategies—the best one is a function of the CSR and NLP algorithmic sophistication and the allowable amount of computation. Finding the best set of likelihood function parameters is an optimization which can only be performed when the components are integrated into a complete SLS.

2.2 Partial Theory Memory

Memoryless CSR and NLP components as used in 1.1 are inefficient because they require recomputation of the embedded left sentence likelihoods. Thus, both the CSR and the NLP will cache the partial theories and the information required to efficiently compute any extensions of those theories. The theory identifiers will have a one-to-one correspondence with the theories.

An alternative would be to store all partial theory information on the stack. This would allow an "almost memoryless" CSR and NLP. This scheme has been rejected for the present, due to its communications overhead. It might be useful in a later version for a loosly-coupled multi-processor environment. (See Sec. 3.2.)

2.3 Stochastic Grammars

Likelihoods (which are, of course, based upon probabilities) are the common language for communication between the two modules and the search control. Thus, grammars which give the probability of a full or partial sentence provide much more information to the combined CSR-NLP system than grammars which just accept or reject a sentence. The simple strategy of estimating the probability of a word as 1/(nr of possible words at this point) may or may not be useful. (It does not help the Resource Management word-pair grammar when used in our CSR.) A much better first cut at a stochastic grammar would be to use N-gram probabilities on top of an "accept-reject" grammar. In the long run, the probabilities should be integrated into the NL grammar, but the first-cut is a reasonable baseline. (Observe, for instance, IBM's success with purely N-gram grammars [1].)

The control scheme used in this proposed specification is tolerant: it can handle full probabilities, branching factor based probabilities, or just acceptance-rejection "probabilities" (i.e., 1's or 0's). Presumably, the more accurate the probabilities, the better the overall performance.

2.4 Fast matches

To reduce the search space, both the CSR and NLP will provide fast matches. These matches take a partial theory and use computationally-efficient methods for providing a quick estimate of the probabilities of the words which may follow. The lists from both components are combined to give the stack a list of words for the slower detailed match. The goal here is just to get the correct word on a small list of candidates. The "fast" probabilities will be used in combining, ordering, and pruning the list, but not in the stack likelihood function.

Methods for performing acoustic fast matches are currently known. NLP fast matches may or may not be possible. (Typically, neither will be available in the early stages of module development.) The interface will still be able to operate, but a wider search and more computation will generally be required.

2.5 Multiple Output Sentences: Top-N Mode

The stack controller can continue to output sentences in decreasing likelihood order. Thus, the user may be asked to choose from a short list of outputs if the system cannot choose one sufficiently reliably.

This mode may also be used to allow non-left-to-right NLP search strategies. The SC-CSR can operate without a grammar or with a purely stochastic grammar (such as N-gram) to generate a list of sentences with (stack) likelihoods. The NLP can then add its likelihood contribution and the best sentence in the list is chosen. (In the case of an accept-reject grammar, the NLP can simply reject non-grammatical sentences in order until one is accepted.) This decoupled mode will reduce the overall computation over the coupled mode only if the NLP requires significantly more computation than the no (or limited) grammar CSR.

This mode may also be used in the tradeoff of search width vs. risk of search error tradeoff. If the search is narrowed too much by increasing α, the sentences may be recognized out of (likelihood function) order. It may be cheaper to run a narrower search and choose the winner later than to run an (empirically) admissible search where the best answer will be output first. Again, these tradeoffs can only be determined in the context of a complete system.

2.6 Second Stage Re-evaluation or Discrimination

If a second stage re-evaluation of the evidence for the top few sentences is desired, the system can be operated in Top-N mode. When the Top-N list is full, a re-evaluation may be performed and the chosen sentence output. This mode only makes sense if a more detailed but greedier or non-left-to-right acoustic matching algorithm or NLP is used. This is similar to the decoupled mode mentioned in 2.5, except (hopefully) more accurate re-evaluation is being performed after the initial evaluation, using the stack. The search then proceeds in three stages: fast-coarse, medium-medium, and slow-detailed.

2.7 Speech Understanding

Since in speech understanding more than one word sequence can have the same meaning, a mechanism has been considered for combining theories. However, such a combination is incompatible with the Top-N re-evaluations described in 2.5 and 2.6. Once two theories are combined, they cannot be separated, and it may be necessary to distinguish between them in the re-evaluation. Thus, such a mechanism is not being included in this version of the interface specification.

The "normal" output of this system is the best word sequence which matches the acoustic and NL constraints. In addition, a mechanism is included for the NLP to output the meaning of the recognized sentence. This meaning will be expressed as text (i.e., ascii characters to make it machine-independent), but its format is undefined by this specification. This will allow the NLP to feed an interpretation or a parse tree to a later module for execution. For example, a database query SLS might output in a database query language or it might output a parse tree for later interpretation into a database query. (Of course, if the SLS is fully integrated into the task, its explicit output might be ignored—its output might be a change of state in the task which may be observable by the user via other modalities. For instance, a chess-playing system might move the chess piece and the user would just see the move on the game board.)

2.8 Features

Linguistic features which have acoustic expression (prosodics, beginning of sentence, end of sentence, etc.) may be attached to words by the NLP. Global features, i.e., features which apply for the entire sentence, must be stated at the beginning of the sentence (due to left-to-right evaluations). A global feature is treated as if it is attached to each and every word. There is a mechanism for the CSR and NLP to exchange feature lists to allow the systems to adapt to each other.

The actual features are undefined by this specification. Only the syntax and mechanisms for transmission are defined here. The features themselves are just text strings—they have no meaning except as interpreted by the CSR and NLP.

2.9 Control

The stack is the sole controller of the system. It sends out a request to a slave and waits for a reply. Either slave (i.e., the CSR or NLP) may, in turn, make a request of a helper, but any such helpers must be slaves of the CSR or NLP. Neither the CSR or NLP nor any helpers may initiate any action involving the stack.

2.10 Integration of the Stack and the CSR

If the system were configured into three separate modules as shown in the figure, it would require excessive communications overhead. The communications with the NLP are simpler than with the CSR—time registration is not an issue for the NLP. Because the CSR must actually return time distributions (likelihood as a function of time), the stack and the CSR are integrated into a single stack-controller CSR module (the SC-CSR) to remove the higher bandwidth channel. This causes no change in the control structure: the stack is still the sole master and the CSR and the NLP are still its slaves. This also causes no change in the NLP interface.

2.11 Search Aborts

To allow efficient "layered" grammars, the NLP may request a search abort. This abort keeps the same acoustic data but re-initializes the stack to its initial state. Thus, a system which first tries a restrictive grammar and then decides that this grammar is unable to match the input, may abort the search and try again with a less restrictive grammar. The NLP may request as many aborts as necessary (although it may be necessary to place an upper limit enforced by the controller to prevent infinite loops).

2.12 Errors

Either the CSR or the NLP can make an error reply to the stack. Four responses are possible: ignore the error, abort this theory, abort this sentence, or abort the program. The first two responses have the option of reporting the error, the third and fourth must report the error. (For instance, in a demo one might wish to suppress error reporting, while in a debugging run, one might want to see all of the errors.) A possible cause for non-fatal errors might be features which are only implemented for some phones in the CSR.

2.13 Comments

Either the SC-CSR or the NLP may place comments onto the pipe interface. These comments will be ignored by the modules. Their only purpose is to place additional information into the communication streams for debugging or demonstration purposes.

3 THE ARCHITECTURE

3.1 The Physical Connection

Logically, the architecture consists of the three parts listed above: the stack controller (SC), a CSR, and an NLP. (As described in 2.10, the stack controller will be combined in a single process with the CSR, but will have the same functionality as if the two were separate.) The SC-CSR process will communicate with the NLP process via UNIX pipes. (A complete interchange has been benchmarked at about 1 ms on a SUN 4/260.) Therefore, the two processes need not be written in the same language and need not even be running on the same machine. (This interchange has been benchmarked at about 4 ms between a SUN 4/260 and 4/110 on our local Ethernet. Network overhead would be prohibitive if any number of gateways were involved.) The NLP will receive its commands on the I/O channel "stdin" and reply on I/O channel "stdout". (Stderr will retain its usual function.)

The specification as defined here, uses standard (unnamed) pipes. An easy way to make inter-machine pipes is with the *rsh* (remote shell) command. (*Rsh* sets up stdin, stdout, and stderr such that the network between the machines is invisible.) An alternative is to use sockets. (Pipes are implemented on some machines using sockets.) Sockets have some advantages, but are more complex to use. Thus any attempt to include sockets in this specification will be delayed until a clear need is developed. Once the socket-based interconnection is established, the communication would be the same as in the pipe-based interconnection.

To minimize the communications overhead, the request for detailed matches may be batched in groups which are extensions of the same theory. Thus the block of commands will be sent from the SC to to the NLP and the replies will be expected as a block (in corresponding order) when the NLP is finished. This will be particularly important when two separate machines are used.

3.2 Parallel Processing

If the CSR and NLP are implemented on separate machines, they may execute simultaneously—i.e., both may perform a fast match for the same theory, or both may perform the (possibly blocked) detailed analysis of a theory.

Parallel execution of the CSR or NLP can be performed by removing several theories from the stack and sending each to a different processor. The difficulty centers on the cached theories which must be located and transmitted between processors on demand. (If all theory information were stored on the stack, the CSR and NLP modules would be memoryless and this would not be a problem. However, all partial theory information would have to be transferred from and onto the stack for every operation. The overhead would be prohibitive.) Only the form of parallelism described in the previous paragraph is supported in this version of the specification. If necessary, a later version could support the second form. Note that the system would eventually bottleneck on the stack controller.

4 THE DATA FORMAT SPECIFICATION

4.1 The Messages

The messages will consist entirely of text in order to make them machine and language independent and easy to debug. (Appropriate use of hashing and special purpose I/O routines can be used to minimize the overhead of conversion to and from text.) Both processes will cache partial theories which will be identified by a positive unordered integer handle (label). Handle "0" will be the null theory. Communications between the stack-CSR and the NLP will be in a command-reply format.

Features are expressed as "word\{feature-value-pair1 feature-value-pair2 ....\}" and global features must be asserted at the beginning of the sentence before any words: "\{global-feature-value-pair1 global-feature-value-pair2\} word1.....". The features are not interpreted in any way by the stack–they are simply passed as (ascii) text between the CSR and the NLP. (**Note**: feature=value will be passed from the NLP to the CSR—it will not be interpreted by the stack.) Word features will override global features which will, in turn, override default features. The actual features are not defined by this specification.

4.2 Data Formats

Most of the messages are short and can be transmitted as a single line terminated by a "new-line" character (i.e., a standard UNIX single line). Lists are transmitted as a group of lines, one list item per line, and terminated by a blank line. White space shall separate items on a line. All probabilities and likelihoods are expressed in log base 10, and log10(0) shall be expressed as "-Inf". The numbers themselves will be written as [-]x.xxx (C language f format).

4.3 Top-N Mode Output Format

Top-N mode will output its sentences in the following format: a likelihood, white space, the sentence text, and a "new line" per sentence. A blank line terminates the output list. The list may or may not be in likelihood order. An ordered list will make further processing more convenient, but an unordered list can be output as soon as each sentence is found to allow parallelism with any later processing.

The list of exchanges is (optional parts are shown as [...], lists are bounded by <...>, and control responses begin with a "\"):

Stack-CSR	NL Reply	Explanation
ready ver-nr	ok	Ready to go, protocol version number
features <feature-list>	<feature-list>	transmit CSR feature list to NLP receive NLP feature list from NLP (lists may be null) (default null lists)
feature-defaults	<feature-list>	request list of feature defaults
fastmatch	yes/no	Does the NLP have a fast match?
reset	ok	Reset NLP to start state
old-id <new-id word list>	<likelihood [\end or \optend] list>	append word to old-id, assign to new-id respond with incremental log-likelihood the old id appears only on the 1st item if(\end) must be end of sentence if(\optend) optional end of sentence
meaning id	text-meaning	give the meaning of the sentence (language of text-meaning undefined)
fast id	<wd likelihood-list>	fast match
purge id	ok	purge [partial] theory
norm id	likelihood	get A* normalization prob for id
(any)	\abort	abort search, restart with same input
(any)	\error react# [explanation]	error return from any command react#=0 ignore following lines are a normal response react#=1 delete theory react#=2 give up on sentence react#=3 abort program if present, the explanation is reported
\# SC-CSR comment	\# NLP comment	A comment from either source has a '#' at the start of the line.

"\" is used to introduce anything which must be interpreted as a control word where it might be confused with a vocabulary word. ("\" itself is written "\\"). All lists are terminated by a blank line.

A typical session for the sentence "who is he" might be (the acoustic probabilities do not show at the interface):

Stack-Controller	NL Reply	Comments
ready 1.1	ok	
features	stress	list of features
(null-list)		no features have been agreed upon
reset	ok	
fast 0	when -1.1 who -3.4 show -2.2 a -2.1	possible first words of sentence
0 1 when	-2.4	theory "when", (batched command)
2 who	-1.3	theory "who"
fast 2	is -2.2 was -3.4	possible extensions of theory "who"
2 3 is	-1.2	theory "who is", (batched)
4 was	-1.5	theory "who was"
fast 3	he -2.0 she -2.0	possible extensions of theory "who is"
3 5 he	-1.0 \end	theory "who is he \end", (batched)
6 she	-1.1 \end	theory "who is she \end"
(stack now picks 5 and outputs "who is he")		
reset	ok	ready for next sentence

5 SIMULATORS

To allow each group to work on its part of the task (CSR or NLP) independently of the other part, a set of simulators will be used. These simulators will communicate using the protocols specified above. Both would be designed to be computationally cheap to expedite the developmental work.

The stack/CSR simulator will be text-driven, use a dictionary and acoustic phoneme models generated from real speech data to cause errors to be "realistic" and have controls to adjust the error rates. (NLP evaluation tests could use defined settings of the control parameters.)

The NLP simulator would use an N-gram language model for efficiency. For the Resource Management database, the BBN word-pair or BBN class grammar could be used.

6 SUMMARY

This specification is an attempt to provide a reasonable set of protocols for integrating CSR and NLP components into a unified structure. We have addressed all issues that we could think of, but undoubtedly, new issues will arise or some of the decisions made in this version will need to be changed. The version numbers were included to provide a coherent mechanism for graceful growth of this interface to meet our future needs.

References

[1] L.R. Bahl, F. Jelinek, and R. L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," PAMI-5, No. 2, March 1983.

[2] N. J. Nilsson, "Problem-Solving Methods in Artificial Intelligence" (McGraw-Hill, New York), 1971.