12-31-1991

Editing digital audio using a perceptual model

J. Mark Goode
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

Editing Digital Audio Using a Perceptual Model

by
J. Mark Goode

In this paper we describe an intelligent system used to aid an audio recording producer in the selection of edit points in a musical sequence. It is possible to make meaningful recommendations concerning auditory input data without the ability to fully classify that data or interpret all outside goals. Perceptual modeling of the auditory input can allow sufficient reasoning ability for an intelligent system to be of use editing digital audio. To create a perceptual model, the information which can be extracted by the ear is embodied in several expert sections which communicate with a rule based decision maker via a blackboard. The envelope expert "listens" for the changes in amplitude. The spectrum expert examines spectral changes. The rhythm expert examines the distances between note crests. This paper is intended to describe the architecture and discuss its use in the creation of an intelligent editing system.

EDITING DIGITAL AUDIO USING A PERCEPTUAL MODEL

by
J. Mark Goode

A Thesis
Submitted to the Faculty of the Graduate Division of the
New Jersey Institute of Technology in Partial Fulfillment of the Requirements for the Degree of
Master of Science
Department of Computer Science
December 1991

APPROVAL PAGE
Editing Digital Audio Using a Perceptual Model

by
J. Mark Goode

Dr. Bonnie MacKellar, Thesis Advisor
Professor of Computer Science, NJIT

_____

_____

BIOGRAPHICAL SKETCH

Author: J. Mark Goode

Degree: Bachelor of Music in Music Engineering Technology

Date: December 1991

Date of Birth:

Place of Birth:

Undergraduate Education:

Bachelor of Music in Music Engineering Technology, University of Miami, Coral Gables, Florida, 1983

Major: Computer Science

Presentations and Publications:

"db Test Report: Goldline 30" *db The Sound Engineering Magazine* 17-2 (1983)

"db Test Report: The Shure AMS Automatic Mixer" *db The Sound Engineering Magazine* 17-6 (1983)

This thesis is dedicated to

Mrs. Marie Goode

Acknowledgement

The author wishes to acknowledge the effort of Dr. Bonnie MacKellar for her supervision and support under unfavorable, to say the least, working conditions of long distance mail.

Additional thanks go to Audio Animation, Inc. for the use of their equipment and facilities.

The author appreciates the suggestions and help concerning the mathematical analysis from David Landau and Gilad Keren from Audio Animation, Inc.

A special thanks is given to Dr. James Geller for his help in finding such a patient advisor in both music and computer science.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

An edit is the selection of a specific point in two digital audio sequences where the first part of the first sequence will be joined to the second part of the second sequence. The process of editing two audio sources is primarily difficult because there is no fixed definition of what constitutes an acceptable edit. An *acceptable edit* for the purpose of this paper is one in which the audible change resulting solely from the edit is not distinguishable from the changes in the music itself. Though musical form does influence the acceptability of an edit, it is not considered in this paper. It is possible to expand this architecture to include musical form in the decisions, but that is future research. The application is further restricted to include only music, so that we can use the patterns which naturally develop in musical passages to add constraints to our search space. This is only partially helpful because there is no fixed definition for music. We therefore define *music* to be only audio sequences which can be represented using conventional music notation, i.e. a staff with notes and a determinable time and key signature. This limits the search space by eliminating the problem of lyrics, speech recognition, and decisions based on other arbitrary sounds.

Editing two musical sequences to form a single output sequence, even after selecting the notes in the music which will border the junction point, usually requires many minutes of repeatedly listening to the same passage. After several attempts at the edit, an acceptable compromise between various attributes may be achieved and the edit is retained. Edits may also be rejected after this effort. A vast majority of commercial recordings are subjected to this treatment. It is time consuming and expensive. By allowing an artificially intelligent assistant to listen to the music and make edit propositions, that time may be substantially reduced.

The process of editing music can still become a non-trivial issue if outside goals are left to the system. It is beyond the scope of this research to consider the purpose of editing music at all, but the general reasons might be to produce a sequence that is longer, shorter, or contains different performances than the sequence or sequences originally recorded. In effect, these purposes represent outside goals which cannot be fully determined by the musical information itself. We alleviate this problem by considering all outside goals to be determined by an outside entity, in this case the producer, while we concentrate on modeling the job of the engineer.

Though the external goals are considered to be a problem left to the producer, it should be noted that in a practical situation, the engineer is usually culturally educated similarly to the producer. The engineer and producer typically practice within the same musical culture, i.e. both might be oriented to classical music, or rap. This commonality of education can cause the engineer to make practical decisions that we will consider the domain of the producer. In practice, our model is roughly equivalent to an American engineer who is totally naive in the ways of far eastern culture editing a tape of Japanese folk music.

The editing process is additionally difficult to automate because it has traditionally been considered an area where musical intuition and artistic talent have been a necessary prerequisite. By using perceptual models of the human ear, and a rule based system to represent the guidelines used by an engineer, we can make relatively good attempts at edits in an effort to assist the producer in his objectives.

CHAPTER 2

SURVEY OF CURRENT LITERATURE

There are two forms of editing music. One form is occurring more and more frequently as technology allows the composer himself to generate a larger and larger percentage of the final musical piece using synthesizers (9) and computer generated music (5). In this form, the information to be edited is the instructions for producing the music, i.e. the synthesizer command sequence. This is a far simpler form of editing for two main reasons. First, production rules for an expert system generated for this purpose would be concerned with musical form and could be used in the second form of editing which will be discussed in a moment. This is necessarily true because there is nothing to be classified or identified at a lower level (the music does not exist as a performance yet). Second, the sequence production system (sequencer) is limited, by definition, to the set of sounds and representations that it can produce, therefore, nothing foreign to the music needs to be considered. These systems are common and there is a wide body of information (5) (11) (12) for the commercially available systems. There have also been the development of languages specifically designed for the expression of music and its form within a computer system (1) (9).

The second form of editing is much more difficult and the research has been more limited. Existing performances must be edited. This may be necessary because the various parts were produced at different locations or at different times. For commercial record album releases, different length versions of the same sequence may be necessary for release on different media. In any case, the decision is made by the producer to edit the music with some particular goal in mind. Form rules could be used if the music is first identified and classified. This involves establishing the location of significant musical events in the sequence and producing a notation on which the rules can be applied (2). Most of the available literature is primarily concerned with this identification and classification of musical landmarks to produce a mapping between the location in the sequence and the musical event. Signal processing techniques have focused on the identification of these constructs and the production of the mapping system (3). The primary goals of previous research has been the process of transcription, or creating a representation of the music in a manner in which the computer can retain information about the musical form. Although this is a tremendous aid in terms of communicating with the human operator to obtain the human's intent, it is not essential to produce helpful assistance to the operator.

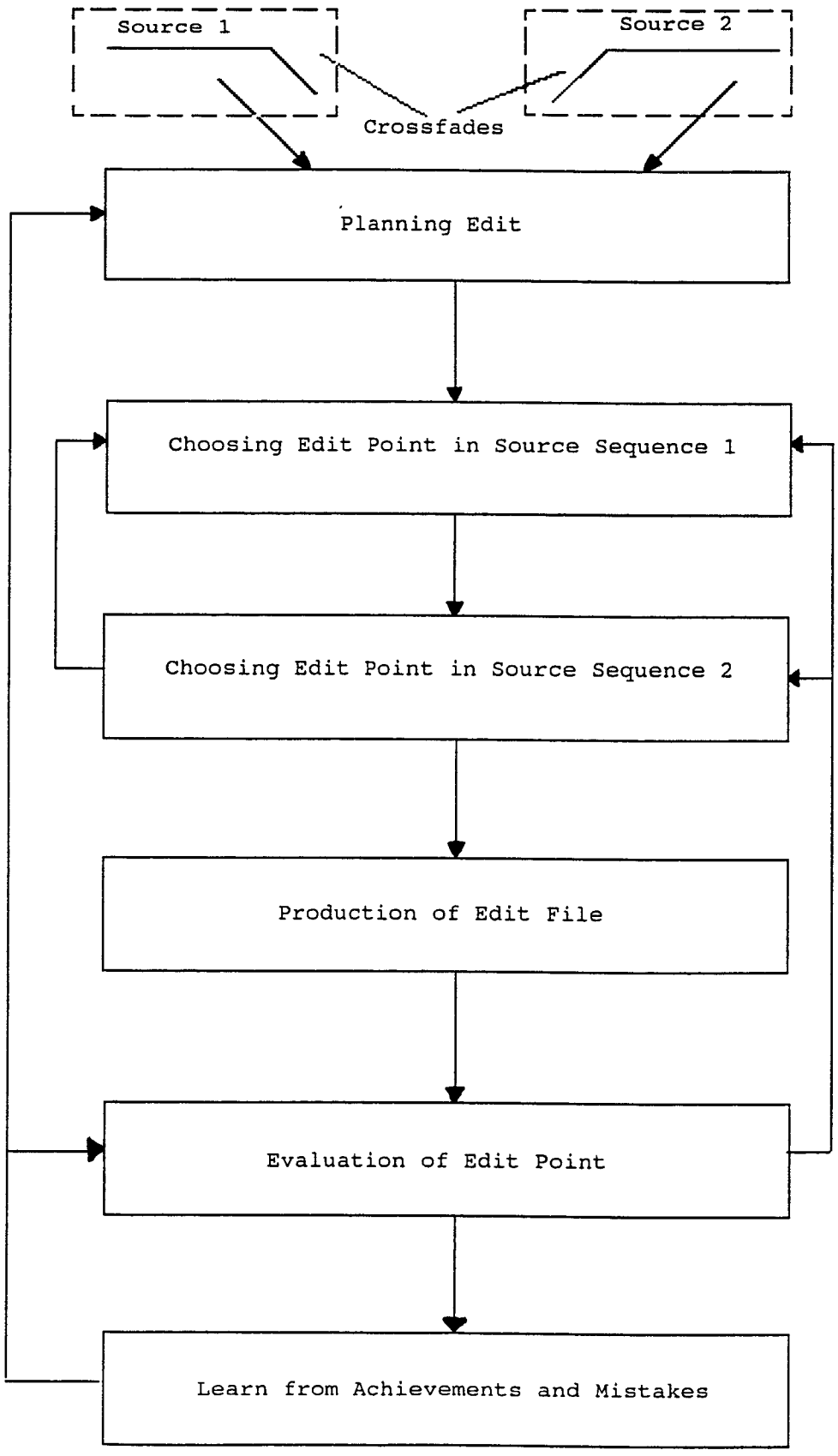

Figure 1. Logical flow of edit process

CHAPTER 3

THE PROCESS OF MAKING AN EDIT

The process of making an edit is shown in figure 1. The edit can be considered the joining of two source sequences at a specified time in each one to produce an output sequence containing the first part of the first sequence and the second part of the second sequence. The two source sequences may be the same sequence. In the case where the source sequences are the same sequence, the output sequence can effectively have duplicated or omitted information.

The first step is to determine which sequences will be used. This implies that the sequences exist as files in a storage medium. The files get to the storage medium by the process of recording. *Recording*, for the purposes of this research, is merely the sampling of the input audio at a specific rate (a compact disk is audio sampled at 44100 times per second) and conversion of the audio to a number representing the deviation of the air pressure from the non-excited norm. The sequence of these samples is then placed in a storage medium, typically a hard disk, for later reproduction.

After the sequences are determined, the approximate time in each sequence where the edit should be performed is determined. It is this stage where classification is most helpful. Classification of the music, in its

simplest form, may be nothing more than the determination of the start and end of various notes. In a more complicated form, it may be the complete construction of a musical score. The segmentation process could provide additional information on musical style which might be used to aid and further constrain the possible edit choices.

In order to obtain an interface with the producer to establish his editing needs, a human engineer, for example, would at least be able to determine that there are notes and discuss the location of potential edits in terms of these notes. To allow an interface between the producer and the automated engineer without the segmentation ability, the range of samples representing the individual notes is manually input into the editing system. This method tends to be cumbersome, but the interface to the producer is not the primary issue.

With human producers and engineers, there is verbal communication to indicate the note that the producer intends to edit around. The interface between engineer, producer, and editing machine, becomes especially blurred in this domain. Traditionally, the engineer and producer are presented with some representation of the musical samples. This is usually an envelope plot of the musical wave form. The degree to which this representation can be abstracted depends primarily on the ability to classify.

As we have said, we assume that an engineer can at least determine notes, and we should represent the music to the producer in the form of classical music notation. Although a complete score does not need to be produced (in fact, the engineer may make his edits without ever hearing the sequences in their entirety), the area around the edit should be represented by a score. This again falls into the area of classification and is avoided in this model. Instead we take the sample which is at the beginning of the note and the sample at the end of the note to represent the range of the note. A more acceptable method of acquiring the producer's intent would be some form of graphical representation, but that is not the primary interest of this research. It should be noted that the addition of classification is necessary if the form of the music is to be considered in the choice of an edit point.

After determination of the input sources and the range in each input sequence where the edit is most desirable, the engineer would listen to the music to try to get a better idea of the structure of the music around the edit range. After picking a possible sample where the edit would take place, the second sequence is auditioned. The most probable point of success is determined for the second sequence and a complete resulting sequence can be proposed. There are several attributes which each

sequence should have to make the final choice for possible edit the most probable to succeed. Such attributes might be a minimum in the envelope of the signal, or an area in which the spectrum contains the least energy. Timing and rhythm are also considerations in choosing a potential edit.

After each sequence has been heard and the most likely point for a successful edit has been selected, the edited sequence is produced and evaluated. The evaluation may look for several areas of failure, again using the same type of information extraction as when the source points were selected. A sudden jump in amplitude attributable to the edit would make the edit unacceptable. A sudden change in the spectral response of the music which is not representative of the attack of a note might also be a reason for rejection. Additionally, the tempo of the music and the rhythm through the edit area cannot be degraded by the edit itself. There are other things which may be cause for rejection of the edit. It is essential in the design of the system, that new criteria and analysis procedures can be easily adapted to the architecture of the editing system.

If the edit passes the evaluation stage, it is presented to the producer as the best solution to the edit that the engineer can produce. The producer has the final say in the edits acceptability, because the

producer is the only one who has the knowledge as to whether the edit meets the criteria for external goals. If the edit does not pass the evaluation stage, potential adjustments are identified, and the original sequences are reauditioned in light of the new information obtained in the joining. The process of edit proposition and evaluation is repeated until either the edit passes the evaluation stage and is presented to the producer or the adjustments are not fruitful and the requested range is categorically rejected as not musically editable. This is a tedious process requiring various lengths of the engineers time depending on the difficulty of the edit. Automated assistance would more quickly eliminate non-productive avenues and aid in selecting a beneficial approach. As the assistance is improved a more complete solution might be obtained.

If the edit was rejected and the producer is able to find an edit in that range which he considers acceptable, it is necessary to determine if the evaluation criteria employed by the engineer was too restrictive or the edit is an exception to normal criteria due to external goals. There is also the possibility that the producer will reject an edit that the engineer has proposed as acceptable. In this case, the evaluation criteria may not be restrictive enough. The other possibility which would cause this is the inability of the model to extract the

information that the producer is using to reject the edit.

This is a learning stage which is closely coupled to the classification stage. It may be necessary to vary the criteria used in evaluation depending on the style of the music and the tastes of the producer. This learning stage involves case based reasoning. It may be that adjustments to the evaluation criteria would correct the false rejection or false acceptance. Due to external goals and varying classifications, changes in evaluation parameters may never make the correct decision for certain cases. In other words, it is not possible for any human being to foretell all artistic decisions that another human being may make. Even if our editing system is perfect in its ability to model a human being, human engineers sometimes make decisions which are unacceptable to the producers that they work with, so we should expect a certain degree of failure.

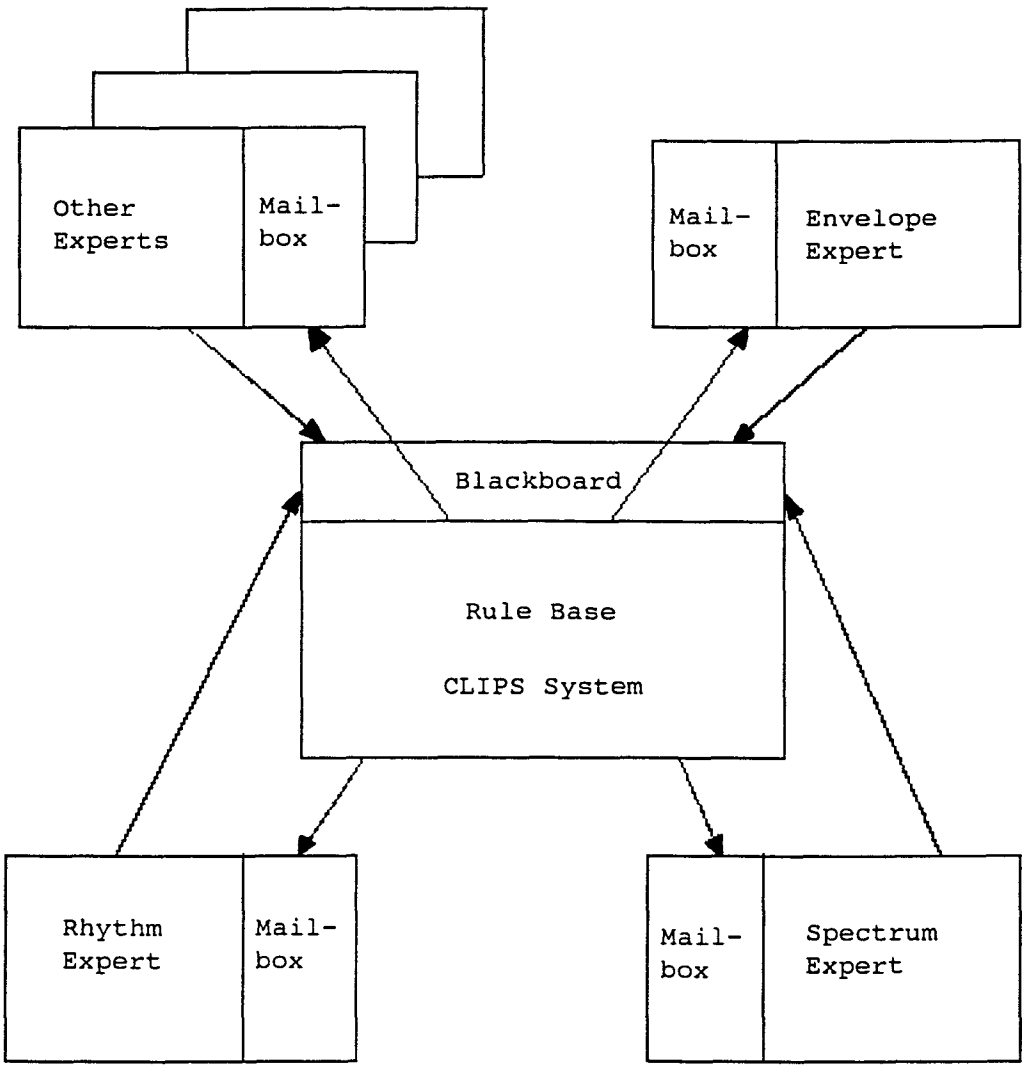

Figure 2. The blackboard interface to the expert tasks. Information flow is in the direction indicated by the arrows.

CHAPTER 4

SYSTEM ARCHITECTURE

The basic system architecture is shown in figure 2. The entire system is written in a real-time operating system which allows tasks to be prioritized. The time-slice features of the operating system allow the system to be later divided among many processors without changing the general architecture and philosophy of the solution. Additionally, slow processes will not block the decision maker from continuing to process results from other tasks. The implementation of the solution must be modular so that if new evaluation techniques are identified, they can easily be incorporated into the existing architecture. The decision maker is a set of rules written in CLIPS (4). These rules represent the ability of the human to "listen for" specific qualities of the music and make decisions based on what is heard. To model the ears, the information is gathered by three experts, though this is a somewhat arbitrary number. Though loudness, spectral content, and rhythm is sufficient for many edits, more experts could be added to increase the abilities of the system. The rules embody the knowledge of when information should be gathered and what significance that knowledge has to the solution.

This knowledge is encapsulated in the CLIPS rule base. CLIPS consists of a set of production rules, a set

of facts, and an *agenda*. If the current facts match the "if" section of a rule, the rule is considered to be *triggered*. Triggered rules are placed on the agenda. As the facts are changed, several rules may be triggered at the same time. After the completion of the analysis of the facts to determine triggered rules, the first triggered rule in the agenda is *fired* and the procedures dictated by the "then" section of the rule are executed. This may be instructions to *assert* new facts, which adds new facts to the fact list, or *retract* existing facts by removing them from the fact list. After each rule is fired, the facts are examined to see if new rules have been triggered or triggered rules should be removed from the agenda.

For the purpose of performing an edit, an objective we wish to achieve is termed a *goal*. In this model, the agenda is used as a *goal stack* where rules evaluate recovered data (facts) and determine if a goal has been met, a goal has transformed into a new goal, or remaining goals must be broken into smaller goals.

As an example, we start with the goal "produce an edit" on the goal stack (agenda). This goal is not directly achievable, but could be achieved if we achieve the goals:

-- Select edit point in first sequence

-- Select edit point in second sequence

```
     -- Join sequences
     -- Evaluate edit.
```

These goals are placed on the goal stack. An example of this type of rule would be:

```
     (defrule get-source-info ""
          (goal plan-edit)
     =>
          (assert (goal get second sampling-frequency))
          (assert (goal get first sampling-frequency))
          (assert (goal get second range))
          (assert (goal get first range))
          (assert (goal get output take))
          (assert (goal get second take))
          (assert (goal get first take)))
```

In this rule, taken from Appendix A, we see that the goal is to plan an edit. In order to plan an edit, we must acquire some information. We post the goals of acquiring each piece of information in the facts.

Eventually goals reach a low level where the intent of the goal must be directly performed. A rule of this type would look like this:

```
     (defrule get-env-edit ""
          (goal get ?take edit-point)
          (envelope take-is ?name)
          (have sampling-rates)
          (envelope range-is ?start ?stop)
```

```
        (?take range-is ?start ?stop)
        (envelope analyzed ?start ?stop)
    =>
        (ask envelope find-edit))
```

In this case, our goal is to get an edit point proposal. If we have all the necessary information, we can ask the envelope expert (a dedicated program discussed later) what that type of analysis would lead us to expect for editing location. The "ask" function is a function attached to CLIPS which will cause the command immediately following the expert's name to be placed in the mailbox of the expert immediately following the "ask". The "ask" syntax is as follows:

(ask <expert name> <command> <parameter> ....)

The mailbox is a separate child program of the expert in question and is discussed in the section about the expert architecture.

The process of evaluating goals is repeated until each objective is either achieved or broken into smaller goals. When the goal stack is empty, the edit has been achieved (if it was possible). This is the manner in which the decision maker decides how to issue commands to the various experts. This example will be continued after the discussion about the various experts.

The entire area of classification and identification of input sequences and ranges for edit points is reduced

to a goal in the agenda. Ideally, a full system would use some graphical interface to obtain the necessary information from the producer. Perhaps a method more appropriate to this model would be to play the sequence for the producer and have him indicate the approximate time where the edit should occur (by pushing a button, for example). It can be argued that the engineer should be able to extract the information where the note indicated begins and ends based on the same type of techniques used to determine the edit point. Indeed, it is a rather simple procedure to determine the start of the notes, however, this is again encroaching on the problem of classification and segmentation (2) (3). The text method that is currently used in this research is cumbersome at best, but does allow the information to be entered.

To accomplish the edit proposal and evaluation stages, a blackboard expert system is used. This is represented in the decision making section by several goals in the rules. The first goal is to determine an edit point for the first sequence. This is then broken into several additional goals that require communication with the individual experts and the extraction of information about the musical sequences they are evaluating. The information that the experts extract is then placed on the blackboard where it is interpreted by

the decision maker. There is no reason that any one expert cannot access information from any other expert, but the present design has the experts only communicating with the decision maker. The decision maker gathers the information from the experts and determines which goals have been met and how events should proceed.

Each of the decision maker and the experts is an individual task. This is to allow the experts to be off-loaded to signal processors as the system expands. For the sake of cost, the signal processing is currently being done on the same computer with the decision maker. The system allows priorities to be assigned to the various tasks so that certain tasks are considered more important than others and are therefore done first. The decision maker could be given priority over the experts and the experts over the blackboard, but all the tasks are currently being run in a time-sharing fashion.

The blackboard is a child process of CLIPS generated when the function "initialize experts" is called from within a CLIPS rule. The "initialize experts" function is another function attached to the CLIPS environment. The function generates a child process for each expert and a child process for the blackboard. Each expert child spawns an additional child process representing the mailbox for that expert.

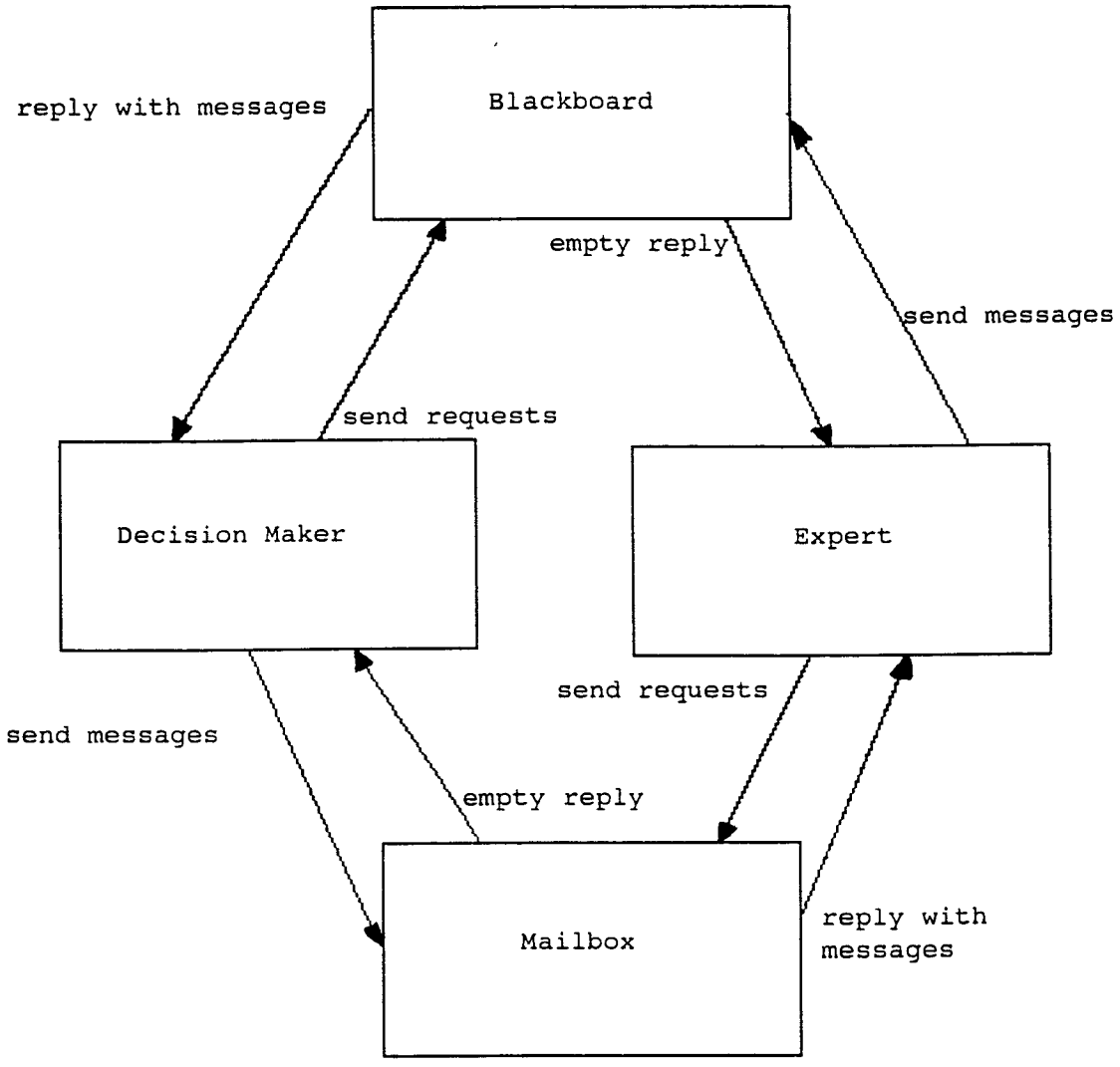

Figure 3. Communication paths between experts and decision maker. Messages are never initiated by the blackboard or mailbox. Handshaking occurs both directions, but information only travels counter-clockwise, preventing deadlock.

The architecture as shown in figure 2 is subject to deadlock if information and messages are allowed to travel in the same direction. If the blackboard is trying to send information to the decision maker, who is trying to send information to an expert, who is posting information on the blackboard, the system will stop. For this reason, the blackboard is never allowed to send messages. The blackboard only replies to received messages. The messages can be instructions to place information on the blackboard or to read information that has been posted there. In the case where the message is an instruction to read the blackboard, the actual movement of information is contained in the reply to that request, allowing the information to move opposite to the direction the message was moving (figure 3). Although there is no security on the system (any task can read the blackboard if it sends the appropriate request) the blackboard is only read by the decision maker. The blackboard is implemented as a message queue.

It is the blackboard expert system which models the perception of the engineer as he edits and is the focus of this research. The evaluation parameters have been set into a single frame and coded directly into the routines. If the classification and learning stages were expanded, additional frames could be added to allow a case based reasoning ability. This would allow some flexibility in

the editor to adapt to the producer who is using the editor and the type of music being edited. The case based reasoning section requires a substantial expenditure of time to produce cases and has not been practical at this point.

A small model of the learning stage is represented by a phase (goal) in the rule based system. The learning stage would need a great deal of expansion to be practical, and, as mentioned before, even the human counterpart we are trying to model cannot expect complete success.

CHAPTER 5

ARCHITECTURE OF THE DECISION MAKING SECTION

The decision making section is constructed of various rules which generally follow the logical progression of an edit shown in figure 1 by using a series of goals or *phases*. As each section of the logical flow is presented, it is put on the CLIPS agenda as a goal which needs to be accomplished. Each of the main goals is too complicated to solve directly so there are rules which break the goal down to smaller and smaller subgoals. Eventually these subgoals can be directly accomplished through one or several commands to the experts.

The blackboard interface is actually one of the attached functions and is not part of a standard CLIPS environment. The blackboard interface requests messages from the blackboard and asserts the reply, providing it is not empty, as if it were a fact asserted by any other rule. Each time the blackboard interface queries the blackboard, either the program will sleep or a fact will be asserted, depending on whether the reply was empty or contained information. The blackboard interface is accessed by sending a request for messages to the blackboard as if it were any other expert. The rule which

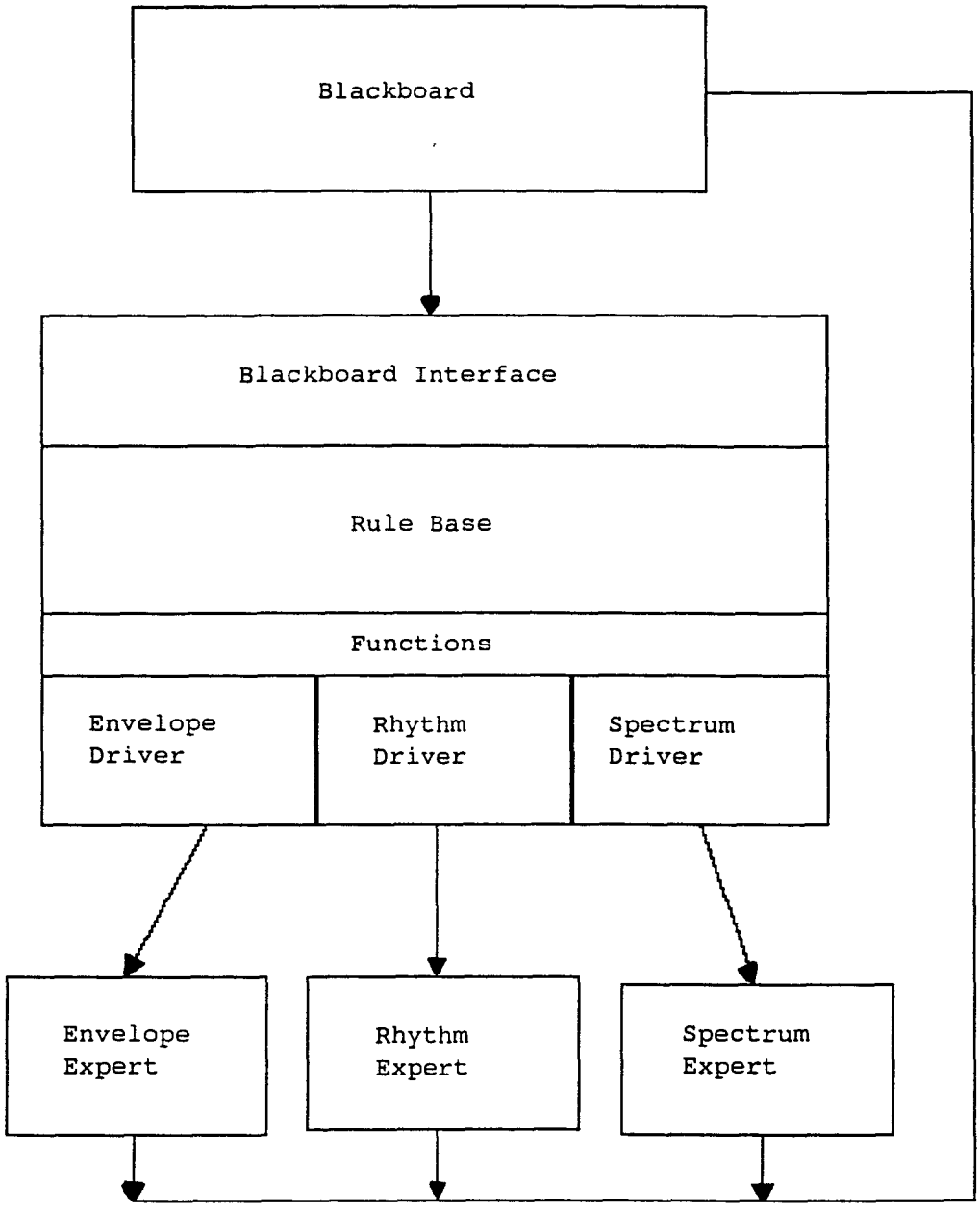

Figure 4. Component parts of decision maker. Information flow is in the direction indicated by the arrows. Functions, drivers and bloackboard interface were non-standard functions linked into the CLIPS system.

causes the blackboard to be queried is:

```
(defrule check-messages ""
     (declare (salience -10))
     ?chk-mssg-fact <- (check-messages)
=>
     (retract ?chk-mssg-fact)
     (ask blackboard)
     (assert (check-messages)))
```

There are no parameters when sending a message to the blackboard since the only appropriate thing to do is ask if there are any messages.

There are two sections which make the decision maker unique to this application (see figure 4). The first section is the interface to the blackboard. One of the few rules which make use of the salience feature of the CLIPS environment is the rule to check the blackboard. *Salience* is the ability to prioritize triggered rules in the agenda. In this case, checking the blackboard is always a triggered rule, however salience is used to assure that the "check the blackboard" rule is only fired when there is no other pending decision. In this design, the blackboard is only checked when no other decisions can be made. If no goals can be accomplished, the decision maker asks the blackboard if there are any messages. To prevent deadlock, the blackboard will immediately respond, whether a message exists or not, to

free the blackboard task so that it can respond when other tasks need to post a message. If there are no messages and no goals which can be accomplished with the current information, it is assumed that the experts are all busy analyzing previous requests. In this case, the decision maker blocks for some time and then repeats the query to the blackboard. This cycle repeats until there is new information from the experts.

The second area that makes the decision maker unique is the expert drivers. For each expert to be used by the decision maker, there is an associated driver routine which parses the messages for that particular expert. The expert drivers are also functions attached to the CLIPS environment, but, like the blackboard, they are accessed by an "ask" syntax in one of the rules. The appropriate driver is then called by the "ask" function. The primary function of the driver is to check rule constructs intended as commands for the experts to assure correct syntax. This is essentially a debug problem. Since each of the tasks is a different program, there is nothing at compile time to prevent illegal messages from being constructed. When all of the experts are running, it is very difficult to determine where the failure occurred when illegal messages are allowed to propagate through the system. The intent of checking syntax is to prevent badly constructed rules from causing an error in one of

the experts and allowing a quick determination that the fault lies in the rule itself. The job of error checking is discussed further below. The second purpose of the expert driver is to construct a message header and message structure from the CLIPS syntax so that message decoding in the receiving expert can be as efficient as possible. By structuring the messages, should the experts be put onto individual processors, the remaining interface kernel can be made extremely small, though this is not of major importance in the way the system is currently being run.

Another use of the salience feature is in the posting of error messages. Should an error message be posted by either one of the expert drivers or by the expert itself, the errors are handled before any other goals. Of course, if the error is contained in a mailbox (or blackboard) with other messages, there is no priority given to that message until it is returned to the decision making section. The message simply maintains its place in the queue as would any other message.

Reasoning that music which is easy for a human to edit should be attempted first, the research has focused on simple edits. Through the tests which have been run up to this point, it is becoming evident that the earlier an edit can be judged inappropriate due to the musical constraints in the edit range, the quicker the response

of the system will appear. Many of the disagreements between experts that were originally intended to be solved by backtracking after both of the edit points had been proposed and evaluation had discovered an error are more efficiently solved by increasing the depth of analysis when the edit range is first examined. Unlike a human being, additional processing can easily be added to the artificially intelligent editor so that more information can be gathered in the early stages of edit proposal. It seems that a more efficient solution lies in pushing the backtracking to the earliest point of edit consideration possible so that repeated access to the musical data is minimized. When editing piano solo passages, for example, it is common to find backtracking in the early stages when frequency information is lacking for the analysis of the envelope. Early attempts at the editor allowed the problem to remain until the evaluation stage, but the repeated analysis of the edit range was slower than forcing a backtrack prior to the evaluation stage. For many of the simpler forms of edits (edits where all three implemented experts can locate a preference edit point and attacks are quick and consistent in style) backtracking from the evaluation stage can be all but eliminated for problems not involving musical form.

There are several additional functions attached to the CLIPS environment. The only other function directly called from a CLIPS syntax is the "make-edit" function which produces an edit attempt file from the two source files. The syntax of a "make-edit" request looks like:

(make-edit <output file> <source1 file>
<edit position 1> <source2 file>
<edit position 2> <sampling rate>)

This function makes use of the "adjust header" function and the "generate crossfade" function, both of which are also attached to the CLIPS environment. All of the functions attached to CLIPS are linked into CLIPS at compile time. The actual function calls made from the CLIPS syntax were intentionally kept to a minimum to make the interface as clean and unobtrusive as possible.

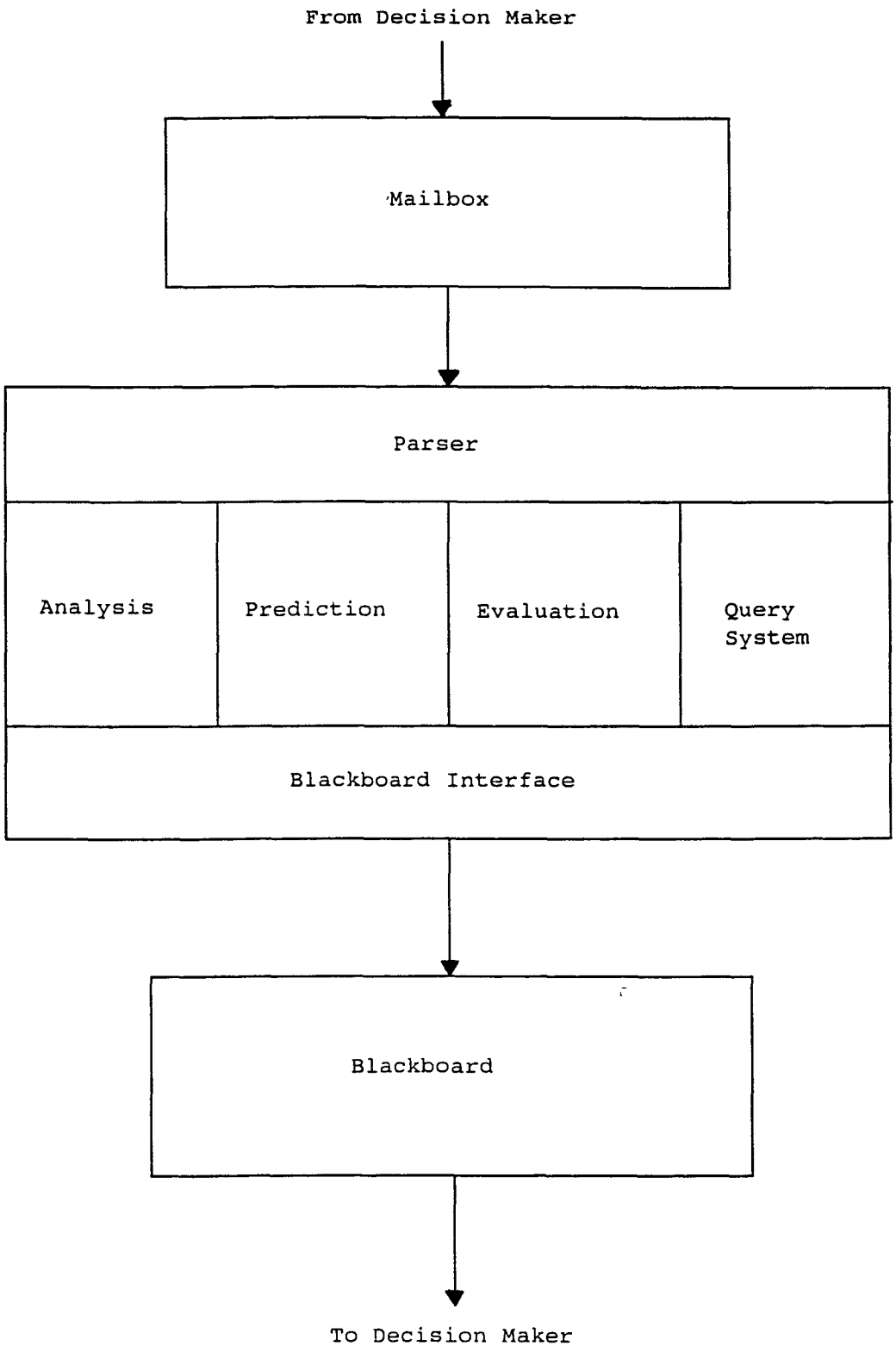

Figure 5. Component parts of an Expert. Information flow is in the direction indicated by the arrows.

CHAPTER 6

ARCHITECTURE OF THE EXPERT SECTIONS

The architecture of an expert section is shown in figure 5. In order to fully prevent deadlock, each expert is given a mailbox. The mailbox is actually the same source code as the blackboard (essentially, the deadlock problem is the same as for the blackboard and decision maker). A mailbox is a message queue whose name is determined by the task that creates it. Upon start up, each expert task generates a mailbox where its messages will be put. The mailbox is actually a completely separate task just as the blackboard. In essence, the blackboard is a mailbox for the decision maker. The primary difference between the blackboard and a mailbox is its use. The blackboard has messages posted by several different expert tasks. The mailboxes are only read by the task that created them and only written to by the decision maker. The primary reason for the mailbox agent (aside from deadlock) was the ability to queue messages so that the decision maker would not be held up waiting for an expert who was still executing the decision maker's last request. By designing the interaction between the decision maker and the experts with the mailbox agent, there should be little difference in the design as the processing becomes more parallel. Parallel

processing results as the experts' analysis sections are moved onto signal processors.

The central routine for each expert is largely a message parser. Each message is evaluated to determine what functions must be executed, and in what order, to accomplish the request received in the mailbox. As each message is parsed, if a response is required, it is constructed in a general buffer which will be asserted as a fact upon receipt by the decision maker. The new fact is posted on the blackboard. If more than one fact needs to be asserted, it must be handled in the function itself. This is usually not necessary. Several of the functions gather additional information, but it is stored in global variables and not released unless specifically requested by the decision maker. For instance, the spectrum expert analyzes the music to determine the lowest frequency present to form its own opinion on edit placement. The information is not offered to the decision maker, however, unless it is specifically requested in the form of a message. Whether it is quicker to gather information while performing some other function or wait until the information is requested depends entirely on the expert and the information involved. The process is generally invisible to the decision maker since the decision maker always continues its operation after

posting each request for information instead of blocking while waiting for that information.

Because all of the experts are running simultaneously, it is often difficult to see what is happening during processing. Most of this information is of little or no use to a user, but is very beneficial in the design stages. To that end, as each message is parsed and each function is called within the decision maker and experts, a notice is posted to an unused operating system console. This console allows the designer to see the experts at work to determine if there are any order specific problems. The messages also serve as "mile markers" to determine what the last thing an expert attempted in case of expert fatalities. The order in which the experts post messages and the contents of those messages varies with the processing power for various tasks and the problem being worked on. An example of this type of output is:

Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Decision maker creating edited file.
Decision maker adjusting header information.
Decision maker is generating a crossfade.
Decision maker completed edit attempt.

Envelope evaluating edit.

Rhythm evaluating edit.

Spectrum evaluating edit point

Envelope accepting edit.

Rhythm accepting edit.

In this example, the production of an edit attempt can be seen by the text from the decision maker. The analysis is started in each of the experts and two of the experts have responded by accepting the edit. If different hardware is used, certain tasks may take a longer time relative to other tasks, and the order of response from the experts will vary.

CHAPTER 7

ENVELOPE ANALYZER EXPERT

One of the existing experts is the envelope analyzer. Its primary responsibility is to make amplitude judgments concerning the input sequences. In general, the most probable points for edit are the lowest amplitude that a note reaches prior to the amplitude increase from the next note.

Amplitude is generally measured by peak to peak or VU (visual unit) meters. These measure the amplitude by keeping a running sum of the absolute value of the samples. A very fast attack and relatively slow decay time are maintained by allowing instant increases in the running sum value, but subtracting a set portion of the value each time a value lower than the current average is obtained. Amplitude measurements allow most note ranges to be determined during classification, since notes often run from a time just prior to an amplitude crest to the next amplitude crest.

Most envelope analysis methods employ a running sum technique where the amplitude of the music is related to the energy over time. The running sum method has definite disadvantages, however, as do most peak to peak meter and VU solutions. The faster the decay time is set, the more these methods tend to follow the wave itself instead of the envelope of the wave. This problem is especially

visible when there is a predominance of low frequencies. If the meter is set fast enough to decay from full amplitude with a high frequency at the rate that the ear decays, the meter will not present a valid amplitude picture for the lower frequencies. Many artificial low points are obtained because the low frequencies may take a significant amount of time to cross the zero point in their oscillations. Instead, a model is used where the lowest frequency to be expected is given to the expert. Instant release times are possible if no additional events occur within one cycle of the lowest expected frequency.

The analysis uses a window large enough to account for the lowest expected frequency. The analysis begins with the last obtained peak (initially zero). Because the algorithm will set the window to start at the point it last found to be a peak, the window start is, by definition, the last recorded peak. The analysis will find the next sample larger than the start of the window. If none is larger, the sample closest in level to the start of the window is used. This effectively allows an instant decay as long as there is no new maximum within the time of one cycle of the lowest frequency.

As an example, suppose the lowest frequency in a given piece of music is thirty hertz. First the sampling rate is divided by thirty to determine the size of the

sliding analysis window. The window is filled with values from the sequence in question, starting with the value representing the beginning of the range. The absolute value of all the values in the window is taken so that excursions on both sides of zero are accounted for. Starting at the beginning of the window, the values are examined. The first time a local maximum is encountered, its value and its location is recorded as the first peak in an envelope recording array. The window beginning is set to the location following this peak. The analysis continues looking for local maximums. If a local maximum occurs which is greater than the last one recorded, its value and location is recorded in the next envelope recording array location, and the window beginning is set to the next location after that local maximum. The process is repeated. If an analysis does not produce a local maximum greater than the last one recorded before reaching the end of the window, the largest local maximum in the window is stored along with its location and the window beginning is set to the location following that. By using this method, both rapid attacks and rapid decays can be tracked without the effect of waveshape tracking occurring. The only remaining problem is when two notes near the lowest determined frequency are near each other and cause amplitude beating. This has not proven to be a problem in the editing examples we have tried so far.

Music generally contains more complex information than two sustained tones, though situations such as this may be solved by agreement from the other two experts.

The envelope analyzer can provide answers to the rate of change in the amplitude near the edit point. In the evaluation stage, sudden changes in amplitude across the edit area which would not have occurred in the original sequence may represent problem areas which should cause the edit to be resubmitted with the selection parameters weighted so that the next appropriate edit point in the range can be tried. This is done by posting a preference containing information on which source sequence would most likely solve the problem and which direction the expert feels the edit should be moved.

At this stage of research, the evaluation section consists of two main comparisons, crest shifts and edit point shifts. If the second crest of the second edit point range is significantly different (determined by the frame value currently inserted directly into the code itself) than the second crest of the first edit point range, the edit point will cause a notable shift in the amplitude through the edit area. This will cause immediate and complete rejection of the edit point region. Since the edit cannot be adjusted to account for this type of change, no backtracking would be helpful.

The rejection statistic is different if the amplitude is decreasing than it is for an increase.

The second problem could cause backtracking. One window is examined on both sides of the edit being evaluated. If the edit point causes an envelope shift based on the resulting maximum values from the two windows, the edit may sound better by making the edit slightly into the note itself. This could be accomplished by shifting the appropriate source edit forward so that part of the next attack would be cut off. This might allow the increase in amplitude from the next note to hide the envelope shift from unequal levels. Of course, as the edit point is shifted, the rhythm will begin to be affected. Eventually two experts will ask for the edit to be moved in opposite directions. This will also cause the edit to be rejected. There are other available options, but this type of backtracking has not yet been fully explored.

CHAPTER 8

THE SPECTRUM EXPERT

The spectrum is analyzed by taking an FFT (Fast Fourier Transform) of the sequence of samples. This is similar to the analysis done for envelope, although it is more extensive. The region where most of the spectral components are at a minimum amplitude or a spectral change is eminent usually represents the place where the edit point will most likely be successful. This can be detected by looking for the high points in the spectrum (the loudest frequencies) and determining if they are increasing or decreasing. A spectral change will produce a new set of frequencies which are loudest. In certain musical sequences, variation in amplitude from the type of instrument used may make amplitude analysis erroneous. In cases such as this, a more appropriate edit point can be found by looking for the area where the spectral content suddenly changes. This can be located by the spectrum analyzer.

In the evaluation stage, the sudden disappearances of the certain frequencies, not representative of the rate of the decay of specific frequencies in the original, may indicate a reason for rejection. The degree to which the differences are significant should be held in the frame representing this particular case. Though

this is static in the current research, it can be made very useful as the classification section is expanded.

As an example, the FFT window is filled with the last values of the range, so that the final range value is the last value of the window. An FFT is done to the window. A maximum value locator fills an array with the location of the maximum values of the spectrum. These should represent the most significant frequencies for this period of time.

The window is then shifted earlier into the music by fifty samples. The FFT of the new window is taken. The routine again looks for the location of maximum values. This time the resulting maximum locations are compared to the previous maximum locations. If any locations are present that did not exist before a counter is incremented. The window continues to shift and the process continues to repeat until the counter reaches an appropriate threshold (fifty changes). At this point, all the frequencies have shifted and the edit point is proposed at the location in the center of the current window.

Current problems with the evaluation section have been caused by two basic areas. Performing Fast Fourier Transforms on a general purpose processors (in this case a Intel 80386 based system) is typically very slow. The tradeoff involved in gathering enough points to make low

frequency information accurate causes the system to be extraordinarily slow. Resolution in the current system is approximately 44Hz. This represents an octave in the bass notes of a simple piano. This has made the evaluation stage on a personal computer almost useless in terms of making suggestions concerning the edit location and direction of possible movement. Many of these problems could be solved by offloading the spectrum analysis to a Digital Signal Processor (DSP).

The second area of difficulty is again associated with the low frequencies. Although FFT analysis is done on windows of 1024 samples, this is only a single wavelength at 44.1Hz. Nonlinear response effects can easily be seen between 44.1Hz and 88.2Hz. Additions of windowing functions such as a Hamming window can help to reduce this problem, but at the cost of additional analysis time. The analysis of the spectrum is already the largest portion of processing time.

In the current implementation of the editor, evaluations are done every 50 samples. These spectral evaluations are compared to the evaluations neighboring the current FFT. Spurious noises can cause the spectrum expert to produce premature edits because they are detected as changes in spectral information. A better result could be achieved at the cost of memory and processing time if the results from each FFT could be

compared to several in each direction to determine for certain whether the spectral content has actually shifted.

It is helpful in segmentation (a classification stage) to subject the music to a high pass filter before doing spectral analysis (3), but this can cause erroneous interpretation of the edit point because the upper partials are mostly non-existent in the areas of time we wish to actually differentiate plausible edit locations from incorrect choices. Problems may also arise when the music in question has very little harmonic content. If the fundamental is filtered out, or even greatly reduced, the information used to determine the best choice for an edit may no longer exist in the time range under examination.

It is difficult with the present implementation to distinguish when the upper partials have fallen below the noise threshold. The spectrum is analyzed in reverse order from the direction of real-time. By doing this, the attack can still be correctly assumed even though it is the noise floor and not the previous decay that causes the spectral shift. To some degree this is not inappropriate since it is common among human engineers to evaluate the musical sequences in both directions (and at several speeds) before choosing an edit point.

CHAPTER 9

THE RHYTHM EXPERT

The rhythm expert examines the length of time between the crest of the previous wave and the edit point. More specifically, the length of time between crests in the original signal should be close to multiples or divisions of two to the length of time between the crests of the resulting signal. The change between crest lengths can be used as a guide for direction should adjustment to the edit point be necessary. If the length of time between crests is quite different from multiples or divisions by two, the evaluation stage may concentrate on this difference as potential reason for rejection. This is especially significant if the original signals have a very distinctive pattern. This information is developed to a large extent in the process of classification, though it is manually mapped to the input sequence in the current examples. It would be possible to analyze only a short section of time, one or two measures, to determine the basic pulse rate of the music. With human performers, some variation in the pulse rate should be expected (2) (3), so analysis outside the immediate range of the edit may be misleading. As with other variations, such as amplitude, rhythmic variations must be compared to surrounding variations to determine if the variation is

introduced solely by the edit, or is within the tolerance of other variations in the music. For both rhythmic and amplitude variations, the focus of the research has been the editing of various performances of the identical section of music. This problem is common in classical productions. In this case, the rhythmic variation can be compared to the variation that existed in both the performances prior to editing. If it is not within the deviation between the two input sequences, the edit should be re-evaluated.

For a brief example, the edit point of the first range is given by the other two experts. This i primarily because no matter where the first edit point is determined to be, the second edit point should be able to be correctly determined base on the first edit point position. Since we know the length of the first note retained by position of the first edit point, and we know the length of both ranges at the time the user selects the second range, we can produce the location in the second edit which will cause the resulting range length to be within the values for the source range lengths.

Rhythm analysis is essential in classification and would play a greater role in the editing process itself if the form of the music is used to determine editability. In the current implementation, it is used primarily as confirmation of the edit selection made by

the other two experts. Rhythm has no input into the selection of the edit point for the first sequence. Given the placement of the first edit point, a very quick determination of likely areas for the second edit point can be produced by the rhythm information. If the musical meter can be assumed invariant across the edit (as it has been in this research), the rhythm information can be used to determine if the next attack has been moved in such a way as to create a shift in the meter. This area has not yet been fully explored in this research. The area of rhythm analysis would benefit greatly from the addition of classification techniques to determine if the alterations in meter are appropriate (as they sometimes are). The sudden introduction of an unusual time signature into a common time passage would almost certainly be an error. In this model, that would be considered a failure on the part of the producer choosing the edit range, though a rhythm expert knowledgeable in the musical form might be capable of preventing such a mistake.

A better, or at least more reliable, method could be developed by extracting more information concerning segmentation. Although it is usually correct to assume that the envelope will be at a maximum at the center of a note crest, this can prove erroneous in certain circumstances where the instrument does heavy modulation

to the amplitude (3). Other forms of segmentation could be given to the rhythm expert to make the determination of rhythm information more independent of the envelope. By relying on the envelope, there will be cases where the rhythm expert will fail because the envelope analysis was incorrect. It would be better if the two experts were more independent of each others information.

An improvement could also be achieved by analyzing segmented crests further than the nearest crest. The current version of the editor is misled if there was a rhythm fluctuation in the notes being edited. This could be improved by analyzing the sequence of notes near the edit point to see if the fluctuation is acceptable.

In this research, the user is expected to enter the classification information by entering the range of samples that represent the note to be edited. This uncouples the rhythm expert from the envelope analysis, however, if the envelope analyzer were to determine note crests for range constraints (in the case where the producer pushes a button to indicate the edit note, for example), this would again become a problem.

CHAPTER 10

ERROR HANDLING

Errors of syntax coming from the goals in the decision maker are handled by that expert's driver in the CLIPS environment. This prevents unnecessary message traffic and attempts to catch the errors at the earliest possible stage. Any errors of this type are actually programming errors in the sense of rule design and are therefore not correctable by the expert even if the message were sent.

The second form of error is the error associated with the input by the user. Sequence files that do not exist, or note ranges not contained in the input files are examples of this form of error. Most of these are currently handled by the rule base system to some degree, though the most appropriate place to handle them would be in a classification or graphics interface stage. The rules which must check for that sort of error become extensive and do not add to the effectiveness of the rule base. Some of these types of errors are handled by the rule base itself, but others are allowed to fall into the third category.

The third category is the errors found by the experts in their attempts to evaluate the information. Any expert may post an error to the blackboard. Although it is given no special attention until it is delivered to

the decision maker, it obtains top priority upon arrival. An error message is constructed by noting that it is an error, who is posting the error, and what the error is. At present, these errors are simply posted as text to the user. In a more elaborate system, the errors could be dealt with by the rule base system in any method acceptable to the type of error.

The syntax necessary for an expert to post an error to the decision maker involves asserting the follow fact syntax:

(error <error condition> <expert registering error>)

This allows the user or designer to immediately locate the expert having difficulty and to determine if the rules are in error or the user has entered incorrect information.

The error system is another example of the use of salience. If an error is registered, it is given top priority. If it is an error which cannot be corrected within the abilities of the rule base, the rule base terminates. The rule which accomplishes this type of error management (or termination in this case) is:

```
(defrule error-handler ""
     (declare (salience 10))
     (error ?type ?by-who)
=>
```

```
        (printout "Error " ?type " was registered by "
             ?by-who)
        (exit))
```

The rule base simply prints the message to the user and terminates processing.

CHAPTER 11

AN EXAMPLE

We left our example before with a goal stack whose next goal is to produce an edit proposition for the first sequence. This could be broken further into:

-- Obtain first sequence

-- Determine sequence sampling rate

-- Acquire area of interest from producer

-- Analyze envelope of area of interest

-- Predict edit based on envelope, etc.

We first select the "Obtain first sequence" goal. This is done by asking the producer what musical passage will be the first section in the completed sequence. This is nothing but a file name. Acquisition of the sequence file is done with Hyperception sampling software (4). A message is then posted in the mailbox of all three experts to set their take names to the first file name. Once the name is confirmed, the goal is removed.

The next goal is to determine the sampling rate. This is done by the envelope expert. A message is posted in the envelope expert mailbox to determine the sampling rate of the first sequence. The answer is written on the blackboard by the envelope expert. The decision maker reads the sampling rate and posts a message to the spectrum analyzer that the sampling rate has been determined at whatever number was written by the envelope

expert. The next objective is to determine the note that is of interest to the producer. This is done by asking the producer to enter the sample number of the beginning of the note and the end of the note. This is not the most effective way, but classification and graphic issues were discussed above. A message is posted to each of the experts informing them of the area which will be under evaluation. Each expert will post a message confirming that the range exists or indicating an error if one was detected. Errors might be that the file does not exist or the range is not contained in that file.

The next goal (assuming no expert posted information such as an error which would alter the goals) is to analyze the envelope of the sequence. This request is posted in the mailbox of the envelope analyzer. When the section has been analyzed, the envelope expert will post a message to that effect on the blackboard. In the meantime, the decision maker may work with the spectrum analyzer to set the sampling rate, take name and, range of interest.

The next goal will be to produce an edit proposal based on envelope. If the spectrum expert has useful information on the blackboard (unlikely at this stage), it may be given to the envelope expert if deemed appropriate by the decision maker. The request to produce an edit prediction is posted in the mailbox for the

envelope expert. When the envelope expert has completed the task, the prediction is placed on the blackboard.

While processing is begun on the first source, the user can be queried for the second source. The second source is checked for validity and its sampling frequency determined. Verification of sampling frequency match must occur before large amounts of processing time are wasted. If the sampling frequencies are mis-matched, an error results and the user is informed. Goals for both sequences may now exist on the goal stack simultaneously. Phases are controlled by rules which check not only that they satisfy a particular goal which is required, but that the information they will need to satisfy it is present.

This process continues until all the goals are met. Exactly what will happen and in what order becomes harder to predict as the process progresses since it is very much dependant on the data contained in the sequences. Timing is also an issue, since each of the experts, each of the mailboxes, the blackboard, and the decision maker are all running independent of each other. The exact order in which various goals are approached depends on the time at which tasks are switched in and out, and the length of time to process each request. The decision maker may begin working on the next goal while the envelope expert is still processing the last request.

An example of the output from the various experts and decision maker is shown in the section about expert architecture (also Appendix G). In the examples, the effects of parallel processing can be seen in the disassociated order in which events are acknowledged.

The experts reflect the phenomenon of "listening for..." particular aspects of the music. The engineer might "listen for" the beginning of the note, or the flute note (a spectrally determinable entity). The engineer may listen to the rhythm to evaluate the edit. In this case, the ears return information to the producer in the way the experts post information on the blackboard. What information is required is determined by the decision maker just as the engineer must decide when and why he needs to listen to the sequence again. The decision maker will "replay" the music through his experts just as the engineer will replay the music to make sure of his edit. The more complicated or difficult the edit, the more frequently the engineer will replay the music (commonly referred to as "scrubbing"). This effect can be seen in the actions of the decision maker as well.

CHAPTER 12

FUTURE RESEARCH

There are several areas in which improvement should be considered. The decision maker rules do not handle backtracking from evaluation very well. This is partly due to the fact that the examples causing backtracking from the evaluation stage could be solved more efficiently by forcing the backtracking to earlier stages of processing. This is done by increasing the data collection abilities of the experts during the phase of initial edit proposition. This is analogous to a novice human editor who frequently overlooks useful information during the edit point selection phase only to realize its usefulness when playing back the completed edit and understanding his mistake. In the test edits that have been done to this point, backtracking in the evaluation stage could easily be solved by collecting more information in the proposition stage. In some ways, this indicates an additional advantage to more complete classification and identification.

The envelope analyzing section should be improved to collect more information concerning rate of degradation. Secondary edit points, which are needed if backtracking does occur (regardless of whether the backtracking is from the evaluation stage or simple disagreement between experts), may not be located close to the first edit

attempt. A more complete messaging system between experts could present information to the envelope expert asking for any secondary edit point, or what the expected degradation would be in a target edit area.

The envelope expert is also written to predict note attacks based on the idea that the attack of the note will be more rapid than its decay. This may be erroneous if the attack is very long in relation to the decay of the previous note, causing the envelope expert to be unable to identify an acceptable edit point. Perhaps a more appropriate technique would be to have several analysis sections within the same expert evaluating the music. The expert itself could be a weighting system to look for consensus among the analysis techniques.

To complicate the issue of envelope analysis, the rule base, and more directly, the edit generation section does not allow changes to the incoming signal amplitude as a whole. In other words, most editors used by human beings (apart from physical editing techniques where a magnetic tape is cut with a razor blade and taped back together) allow the operator to increase or decrease the level of the incoming signal. Some edits which should otherwise be rejected may be compensated for in this manner. Without classification techniques and additional rhythm support, it is not possible to differentiate between edits which need level control and edits which

are not appropriate because of their amplitude incompatibility.

The addition of incoming signal level control falls into a basic category of controls not available on this version of the editor. An additional control which may make certain rejected edits possible is the ability to change crossfade time. In the present implementation, the crossfade time is fixed at ten milliseconds. This is a typical setting, but some edits which cannot be done with this crossfade time become possible with extended crossfade times. An example of this is an edit between live performances where the edit is to be done in the applause. Of course, long crossfade times complicate the problem of rejecting an edit due to shifts in spectrum or amplitude because crossfades of one second or more span enough time that the music itself may be responsible for the shift and the shift would be totally appropriate to the musical context.

The spectrum analysis section could be improved by examining a larger area to determine if the spectrum has actually shifted or there is extraneous noise. The addition of windowing functions may make information at the lower frequencies more palatable to the analysis section. The analysis is often fooled in low frequency information because of the fluctuation in level due to the sliding window phenomena.

An entire class of edits has been eliminated by excluding speech. A class of edits similar in detection (although quite different in classification and representation) are those edits occurring between musical compositions. These edits should be some of the more simple edits to complete, but the current system does not account for those types of problems. The primary difficulty is understanding when the edit is of this type so that alterations can be made to the rhythm detection expert. In a case such as this, the whole objective may be to change the length of time between the last crest of the preceding passage and the first crest of the incoming passage. The current implementation would reject this type of edit. Complete suspension of the rhythm detection and rejection would be an incomplete solution because many times the crest of the note for the incoming passage is timed so that the rhythm is maintained from the preceding passage until the new rhythm can be established. A similar situation occurs when the edit point is between two notes where the note itself has decayed into the noise floor. This problem is approached in the current implementation by evaluating the spectrum backwards in time (from the second crest to the first). It is solved in the envelope analyzer by looking for the increase in envelope slope which represents an attack.

This method has the drawback mentioned above when the attack for a note is long in relation to the decay.

All of the above are rather minor modifications to the existing system. There are three suggestions which would represent major changes. The first is a study on the quantity and types of edits being done. It is unclear how well the current editor solves the editing needs because it is not clear what types of edits are most frequently attempted. This could be a difficult study to complete because the answers received will depend largely on the people asked. The needs in a classical music establishment may be completely different from those in a "jingle house" where the primary product is musical passages less than one minute long. Many of the potential problem areas discussed above may actually be rare circumstances in terms of real problems and the solution as it exists may already be quite useful.

The second area of major change is the incorporation of the classification and identification stage. Although useful decisions can be made without this area, it would no doubt be of great benefit. In all commercial circumstances, the engineer who is doing the edits for the producer can at least identify the notes in the music and where they start and stop. If for no other reason, a more efficient communication between man and machine could be developed so that a more natural interface could

be used. Trying to enter the sample number which maps to the note for the edit is not a natural interface and is cumbersome at best. Significant advantages could be realized by the incorporation of rules exploiting the information available from the classification of the music it terms of rhythm and additional control. Form analysis could easily identify the problem mentioned above where the edit must take place in areas between musical passages and not in the passages themselves.

The final area of improvement is the addition of the learning section. In the current implementation, criteria used to judge rejection or acceptance of the edit is put directly into the expert code. It would be much better to send this information to the expert based on the preferences of the operator. A frame could be developed which holds the criteria for rejection based on the operator's critical nature and the type of music being edited. This may allow even the current implementation a wider range of operation. Other areas of preference which could be included in the operators preferences would be a general time shift. For any particular operator and any particular music type, the operator may prefer the edits to be done slightly earlier in time than the analysis would indicate (or later in time for that matter). This information could also be incorporated into the case based frame system. The current implementation has an

extremely limited version of this where no actual changes to the frame occurs and the frame is directly imbedded into the source code. The only indication is a section which queries the user to see if the edit agreed with the user's expectations. This section could be greatly expanded.

CHAPTER 13

CONCLUSION

Musical passages can be analyzed to produce useful decisions about the music without fully classifying the music. Perceptual models of the human ear can be used to extract similar information to that which an engineer uses to produce an edit. The perceptual models extract information driven by the needs of the decision maker rather than the decision of classification being driven by the perceptual information. Although classification is very useful, it is not necessary to begin the process of making useful decisions. Classification can be the result of information gathering driven by a productive decision and hence aid the decision rather than the classification being the objective itself. The learning process changes parameters by which editing judgments are made, but learning does not have an effect on the method in which editing is done in this model.

The editor is currently capable of simple edits where there is a wide dynamic range in the notes. The primary limitation is in the ability of the editor to vary a sufficient number of variables to achieve more subtle editing talents. For example, the crossfade length is currently fixed, and it is assumed that there is no control of amplitude possible. The parameters which determine when an edit cannot be made are also fixed,

where a case based learning system would be a more appropriate solution. Though there have been only a few examples run through the editor, it would appear that the editor is at least as efficient (though not yet as versatile) as its human counterparts.

APPENDIX A

The Decision Maker

```
/*******************************************************
*
*                          GLOBALS.H
*
*******************************************************
*
*  These are the globals to the user defined functions
*    in the file main.c.  They belong to the functions
*    that interface the decision maker to the experts
*    and the blackboard.
*
*******************************************************/
/*
** Directory where all data is located.
*/

char directory_gca[256];

/*
** File name for output text.
*/

FILE *text_out_gfh;

/*
** This is the information for the blackboard where the
**   messages from experts are posted for use by the
**   clips system.
*/

#define BLAKBORD_FILE          "/editor/bin/blakbord"
#define BLAKBORD_NAME          "blakbord"

/*
** This is the name of the file containing the
**   executable for the envelope analyzer; one of the
**   experts.
*/

#define EXP_ENV_FILE           "/editor/bin/exp-env"
#define EXP_ENV_NAME           "EXP-env"

/*
** This is the name of the file containing the
**   executable for the spectrum analyzer; one of the
**   experts.
*/
```

(globals.h continued)

```
#define EXP_FFT_FILE            "/editor/bin/exp-fft"
#define EXP_FFT_NAME            "EXP-fft"

/*
** This is the name of the file containing the
executable **  for the rhythm analyzer; one of the
experts.
*/

#define EXP_RHY_FILE            "/editor/bin/exp-rhy"
#define EXP_RHY_NAME            "EXP-rhy"

/*
** This number is used by the system to locate the
**   running task of the experts and blackboard. It is
**   used in sending and replying to messages.
*/

int       blakbord_gi;
int       exp_env_gi;
int       exp_fft_gi;
int       exp_rhy_gi;

/*
** Decision makers name
*/

#define   DECISION_NAME       "Decision"

/*
**   Switches for debugging the experts
*/

/*
#define DBG_ENV
#define DBG_FFT
#define DBG_RHY
#define DBG_BLAKBORD
*/


/*
** Global message area
*/

assert_mat                    assert_gma;
```

```
/*******************************************************
*
*                          PROTOS.H
*
*******************************************************
*
*  These are the prototypes to the functions in the
*    clips environment.  Some of them are for functions
*    define in CLIPS and used in the user defined
*    functions.
*
*******************************************************/
/*
** prototypes taken from the 'sysfun.c' file
*/
   int    print_num(char * fileid, float number);

/*
** prototypes taken from the 'sysdep.c' file
*/
   int    init_clips(void);

/*
** prototypes taken from the 'usrint.c' file
*/
   int    command_loop(void);

/*
** clips prototypes used in user defined functions
*/
   float  numget (struct test * test_ptr,
                  char * fun_name);

/*
** prototypes for user functions.
*/
   void   adjust_headers_vf (FILE **take1_fhp,
                             FILE **take2_fhp,
                             FILE **dest_fhp,
                             long   take1_edit_l,
                             long   take2_edit_l);

   void   make_crossfade_vf (int buffer1_ip[],
                             int buffer2_ip[],
                             int dest_buffer_ip[],
                             int crossfade_length_i);


   int    exit_clips (void);
```

```
/*******************************************************
*
*                          USERDEFS.H
*
*******************************************************
*
*  These are the define values and constants used in
*    user functions in the clips environment.
*
*******************************************************/
/*
** These are the key letters used in command. When an
**   ask command is sent, it must begin with one of
**   these characters for the expert's name.
*/
#define BLAKBORD          'b'
#define ENVELOPE          'e'
#define SPECTRUM          's'
#define RHYTHM            'r'

/*
** These are the specifics for creating a HyperSignal
**   file. HyperSignal is a package by Hyperception
**   Inc.
*/
#define HEADER_SIZE           10
#define AMPLITUDE             0
#define FRAMESIZE             1
#define SAMPLING_FREQ_R       2
#define FFT_ORDER             3
#define NUM_DATA_R            4
#define FRAME_OVERLAP         5
#define DATA_TYPE             6
#define USER1                 7
#define SAMPLING_FREQ_D       8
#define NUM_DATA_D            9

/*
** These the possibilities when checking the number of
**   parameters in a CLIPS command.
*/
#define EXACTLY       0
#define AT_LEAST      1
#define NO_MORE_THAN  2

/*
** This is the size of the buffer (in integers) which
**   is used to copy sources to destination when
**   actually making an edit.
*/
#define XFER_BUFFER_SIZE 30000
```

```
/*******************************************************
*
*                          MESSAGES.H
*
*******************************************************
*
* These are the message structures and associated
*    defines for all the expert and blackboard
*    communications. This file is shared by all
*    experts, blackboard, decision maker, and
*    mailboxes. Also included are the defines which are
*    necessary in all programs and experts and need to
*    be changed in all cases at the same time. These
*    defines are primarily used to set environment
*    parameters in case programs are off loaded to
*    other processors.
*
*******************************************************/
/* COMMON DATA AREAS SHARED BETWEEN ALL PROGRAMS --- */

#define TEXT_OUTPUT                "$con2"
#define DATA_DIRECTORY             "/editor/data/"
#define NAME_SERVER_NODE           0

/* MESSAGE IDENTIFICATIONS */
#define ID_EMPTY_REPLY             0
#define ID_SET_RANGE               1
#define ID_SET_LOW_FREQUENCY       2
#define ID_REQUEST_MESSAGE         3
#define ID_EMPTY_QUEUE             4
#define ID_ASSERT_MESSAGE          5
#define ID_GET_SAMPLING_RATE       6
#define ID_SET_TAKE                7
#define ID_ANALYZE                 8
#define ID_FIND_EDIT               9
#define ID_GET_LOW_FREQUENCY       10
#define ID_SET_SAMPLING_RATE       11
#define ID_EVALUATE_EDIT           12
#define ID_SET_EDIT_TO             13
#define ID_SET_TAKE_NUMBER         14

#define ID_TERMINATE               255
#define MAX_MESSAGE_LENGTH         256
```

(messages.h continued)

```
/*
** MESSAGE STRUCTURES -- All structures have an
**   identification byte as the first byte in the
**   message.
*/

typedef struct
{
     char      struct_id_c;
     long      start_of_range_l;
     long      end_of_range_l;
} set_range_mat;

typedef struct
{
     char      struct_id_c;
     char      take_name_ca[17];
} set_take_mat;

typedef struct
{
     char struct_id_c;
     int       low_frequency_i;
} set_low_frequency_mat;

typedef struct
{
     char struct_id_c;
     long rate_l;
} sample_rate_mat;

typedef struct
{
     char struct_id_c;
} id_only_mat;

typedef struct
{
     char struct_id_c;
     char assert_mssg_ca[MAX_MESSAGE_LENGTH-1];
} assert_mat;

typedef struct
{
     char      struct_id_c;
     char      take_name_ca[64];
     long      start_of_range_l;
     long      end_of_range_l;
} evaluate_edit_mat;
```

(messages.h continued)

```
typedef struct
{
     char struct_id_c;
     long point_l;
} set_edit_to_mat;
```

```
/*******************************************************
*
*                          MAIN.C
*
*******************************************************
*
* This is the main routine for the user section of
*    CLIPS. It contains a program for initializing the
*    expert tasks and a program for parsing messages to
*    those tasks. The parser is only responsible for
*    ascertaining which expert the message is for.
*    There is an additional driver for each expert used
*    to parse the messages for its own expert.
*
*******************************************************/
/*
NOTE : fprintf and assert strings have been
artificially split onto two or more lines in many cases
to facilitate reading. This applies to all programs
printed in the appendices.
*/

/*
** These are library files contain in the Computer
**   Inovations Inc C Compiler.
*/

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <magic.h>
#include <taskmsgs.h>
#include <systids.h>

/*
** This is a prototype file for CLIPS functions.
*/
#include "clips.h"

/*
** These are user defined files (see above).
*/
#include "userdefs.h"
#include "protos.h"
#include "messages.h"
#include "globals.h"
```

(main.c continued)

```
main()
  {
   init_clips();
   cl_print("wclips",
     "         CLIPS (V4.10 10/05/87)\n");
   command_loop();
   exit_clips();
  }

/*
** USER FUNCTIONS DEFINED IN THIS MODULE
*/
     int       initialize_experts (void);
     int       ask (void);

/*
** EXTERNAL CLIPS FUNCTIONS REFERENCED IN USER
**   FUNCTIONS
*/

     extern int arg_num_check(char * fun_name,
                              int check_val,
                              int exp_num);
     extern struct values *arg_type_check
                          (char * fun_name,
                           int arg_num,
                           char * exp_type);
     extern struct fact *assert(char * str);

/*
** EXTERNAL USER FUNCTIONS REFERENCED IN THIS MODULE
*/
     extern    int  load_msg_blakbord_if (void);
     extern    int  load_msg_exp_env_if (void);
     extern    int  load_msg_exp_fft_if (void);
     extern    int  load_msg_exp_rhy_if (void);
     extern    int  make_edit (void);
```

(main.c continued)

```
/*******************************************************
*******************************************************
* inititialize_experts () starts the expert tasks
*    running in the background
*******************************************************
*******************************************************/

int       initialize_experts ()

{
     extern    char directory_gca[];

     char timing_c = 0;
     int  send_results_i;

/*
** Start data output file.
*/
     text_out_gfh = fopen (TEXT_OUTPUT,"w");
     fprintf (text_out_gfh,"\nStarting experts");


/*
** Initialize data directory name.
*/
     strcpy (directory_gca,DATA_DIRECTORY);

/*
** Register name for decision maker.
*/
     if (! (name_attach (DECISION_NAME,My_nid)))
     {
          fprintf (text_out_gfh,
               "\nDecision maker. \n   I can't attach
               my name.");
          assert ("error registering-name
               decision-maker");
     }

/*
** Start blackboard task. Make sure that the queue is
**   empty. This also insures that the blackboard name
**   is registered before starting the experts.
*/
#ifdef DBG_BLAKBORD
     fprintf (text_out_gfh,
          "\nBlackboard manager here --- ");
     fprintf (text_out_gfh, "%s%s",
          "\n     Start black board manually",
          " within 15 seconds.");
```

(main.c continued)

```
     if (0 >= (blakbord_gi =
          name_locate (BLAKBORD_NAME,
               NAME_SERVER_NODE,MAX_MESSAGE_LENGTH)))
     {
          fprintf (text_out_gfh,
               "\nHi, Decision maker --");
          fprintf (text_out_gfh,
               "\n     I can't find the blackboard.");
          assert ("error finding-blackboard
               decision-maker");
     }
#else
     if (0 >= (blakbord_gi =
          createq (1,BLAKBORD_FILE,NULL)))
     {
          fprintf (text_out_gfh,"\n%s\n%s",
               "CLIPS USER ERROR",
               "I can't start the blackboard.");
          assert ("error starting-blackboard
               decision-maker");
     }
#endif

/*
** Clear the blackboard by sending a request message
**   This block this program until blackboard manager
**   is running well enough to reply.
*/
     assert_gma.struct_id_c = ID_REQUEST_MESSAGE;
     send (blakbord_gi,(void *) &assert_gma,
          &assert_gma, sizeof(assert_mat),
          sizeof(assert_mat));

/*
** Start envelope expert. This is done by creating a
**   task which will run concurrently with the decision
**   making section. There are two modes which it can
**   be started in, manual or automatic depending on
**   the DBG switch
*/
#ifdef DBG_ENV
     fprintf (text_out_gfh,"\n%s\n%s",
          "Decision maker here --- "
          "Start envelope expert in manually within 15
          seconds");
     sleep (15);
     if (0 >= (exp_env_gi =
          name_locate (EXP_ENV_NAME,NAME_SERVER_NODE,
               MAX_MESSAGE_LENGTH)))
     {
```

(main.c continued)

```
          fprintf (text_out_gfh,
               "\nHi, Decision maker speaking.");
          fprintf (text_out_gfh,
               "\n  I can't locate the manual envelope
               expert.");
          assert ("error
               locating-manual-envelope-expert
               decision-maker");
     }
#else
     if (0 >= (exp_env_gi =
          createq (1,EXP_ENV_FILE,NULL)))
     {
          fprintf (text_out_gfh,
               "\nCLIPS USER ERROR \n  I can't start
               the envelope expert");
          assert ("error starting-envelope-expert
               decision-maker");
     } else {
     /*
     ** Make sure expert has registered its name before
     **   proceeding. In this case, message contents
     **   will never be checked. This is for timing
     **   purposes.
     */
          assert_gma.struct_id_c = ID_REQUEST_MESSAGE;
          send_results_i = send (exp_env_gi,
               (void *) &timing_c,&timing_c,1,1);

     /*
     ** If the task has replied, it should have started
     **   the mailbox. Get the mailbox address.
     */
          if (0 >= (exp_env_gi =
               name_locate (EXP_ENV_NAME,
               NAME_SERVER_NODE, MAX_MESSAGE_LENGTH)))
          {
               fprintf (text_out_gfh,
                    "\nDecision maker --\n  Envelope
                    expert died before receiving
                    message.");
          }
     }
#endif
```

(main.c continued)

```
/*
** Start spectrum expert. This is done by creating a
**   task which will run  concurrently with the
**   decision making section. There are two modes which
**   it can be started in, manual or automatic
**   depending on the DBG switch
*/
#ifdef DBG_FFT
     fprintf (text_out_gfh,
          "\nDecision maker here --- \n Start spectrum
          expert in manually within 15 seconds");
     sleep (15);
     if (0 >= (exp_fft_gi =
          name_locate (EXP_FFT_NAME,NAME_SERVER_NODE,
          MAX_MESSAGE_LENGTH)))
     {
          fprintf (text_out_gfh,
               "\nHi, Decision maker speaking.");
          fprintf (text_out_gfh,
               "\n  I can't locate the manual spectrum
               expert.");
          assert ("error
               locating-manual-spectrum-expert
               decision-maker");
     }
#else
     if (0 >= (exp_fft_gi =
          createq (1,EXP_FFT_FILE,NULL)))
     {
          fprintf (text_out_gfh,
               "\nCLIPS USER ERROR \n  I can't start
               the spectrum expert");
          assert ("error starting-spectrum-expert
               decision-maker");
     } else {


     /*
     ** Make sure expert has registered its name before
     **   proceeding. In this case, message contents
     **   will never be checked. This is for timing
     **   purposes.
     */
          assert_gma.struct_id_c = ID_REQUEST_MESSAGE;
          send_results_i = send (exp_fft_gi,
               (void *) &timing_c,&timing_c,1,1);
```

(main.c continued)

```
          if (exp_fft_gi != send_results_i)
          {
               fprintf (text_out_gfh,
                    "Decision maker got invalid
                    starting return from spectrum
                    expert");
          }

     /*
     ** If the task has replied, it should have started
     **   the mailbox. Get the mailbox address.
     */
          if (0 >= (exp_fft_gi =
               name_locate (EXP_FFT_NAME,
               NAME_SERVER_NODE, MAX_MESSAGE_LENGTH)))
          {
               fprintf (text_out_gfh,
                    "\decision maker --\n
                    Spectrum expert died before
                    receiving message.");
          }
     }
#endif

/*
** Start rhythm expert. This is done by creating a task
**   which will run concurrently with the decision
**   making section. There are two modes which it can
**   be started in, manual or automatic depending on
**   the DBG switch
*/
#ifdef DBG_RHY
     fprintf (text_out_gfh,
          "\decision maker here --- \n Start rhythm
          expert in manually within 15 seconds");
     sleep (15);
     if (0 >= (exp_rhy_gi =
          name_locate (EXP_RHY_NAME,NAME_SERVER_NODE,
          MAX_MESSAGE_LENGTH)))
     {

          fprintf (text_out_gfh,
               "\nHi, Decision maker speaking.");
          fprintf (text_out_gfh,
               "\n  I can't locate the manual rhythm
               expert.");
          assert ("error locating-manual-rhythm-expert
               decision-maker");
     }
```

(main.c continued)

```
#else
     if (0 >= (exp_rhy_gi =
          createq (1,EXP_RHY_FILE,NULL)))
     {
          fprintf (text_out_gfh,
               "\nCLIPS USER ERROR \n  I can't start
               the rhythm expert");
          assert ("error starting-rhythm-expert
               decision-maker");
     } else {
     /*
     ** Make sure expert has registered its name before
     **   proceeding. In this case, message contents
     **   will never be checked. This is for timing
     **   purposes.
     */
          assert_gma.struct_id_c = ID_REQUEST_MESSAGE;
          send_results_i = send (exp_rhy_gi,
               (void *) &timing_c,&timing_c,1,1);
          if (exp_rhy_gi != send_results_i)
          {
               fprintf (text_out_gfh,
                    "Decision maker got invalid
                    starting return from rhythm
                    expert");
          }

     /*
     ** If the task has replied, it should have started
     **   the mailbox. Get the mailbox address.
     */
          if (0 >= (exp_rhy_gi =
               name_locate
                    (EXP_RHY_NAME,NAME_SERVER_NODE,
                    MAX_MESSAGE_LENGTH)))
          {
               fprintf (text_out_gfh,
                    "\decision maker --\n  Rhythm
                    expert died before receiving
                    message.");
          }
     }
#endif


     assert ("initialized experts");
     return (0);
}
```

(main.c continued)

```
/*******************************************************
*******************************************************
*
* This function determines which expert is being
*    queried or commanded and calls his message driver.
*
*******************************************************
*******************************************************/

int       ask ()
{
     char           task_name_ca[17];
     int            return_value_i = 0;
     int            task_id_gi;
     struct values  *arg_ptr;

/*
** Check to see that there are at least two arguments.
**   They are
**        task (expert) to be asked     -- string
**        question to be asked of task  -- string
**        parameter list                -- floats
*/
     if (arg_num_check ("ask",AT_LEAST,1) == -1)
     {
          return (0.0);
     }

/*
** If the name of the expert is not a string, return
**   the value 0.0
*/
     if ((arg_ptr =
          arg_type_check("ask",1,WORD)) == NULL)
     {
          return (0.0);
     }

/*
** Assign the argument to the name of function
*/
     strcpy (task_name_ca,arg_ptr->value);

/*
** Get id of task to receive this message
*/
     switch (*task_name_ca)
     {
```

(main.c continued)

```
        case BLAKBORD :
             task_id_gi = blakbord_gi;
             if (load_msg_blakbord_if ())
             {
                  assert ("error
                       loading-blackboard-message
                       decision-maker");
             }
             break;
        case ENVELOPE :
             task_id_gi = exp_env_gi;
             if (load_msg_exp_env_if ())
             {
                  assert ("error
                       loading-envelope-
                       expert-message
                       decision-maker");
             }
             break;
        case SPECTRUM :
             task_id_gi = exp_fft_gi;
             if (load_msg_exp_fft_if ())
             {
                  assert ("error
                       loading-spectrum-
                       expert-message
                       decision-maker");
             }
             break;
        case RHYTHM :
             task_id_gi = exp_rhy_gi;
             if (load_msg_exp_rhy_if ())
             {
                  assert ("error
                       loading-rhythm-expert-message
                       decision-maker");
             }
             break;
        default :
             assert ("error locating-target-task
                  decision-maker");
             break;
    }

    return (return_value_i);
}
```

(main.c continued)

```
/*********************************************************/
usrfuncs()
  {
     define_function("ask",'i',(int (*)()) ask,"ask");
     define_function("initialize-experts",'i',
          (int (*)())initialize_experts,
          "initialize-experts");
     define_function("make-edit",'i',(int (*)())
          make_edit,"make-edit");
  }
/*********************************************************/

exit_clips ()

{
     fclose (text_out_gfh);
}
```

```
/*******************************************************
*
*                          MSSGBLBO.C
*
*******************************************************
*
* This is the driver for the blackboard itself. It
*    parses messages intended to query the blackboard
*    to see if messages have been posted by the
*    experts.
*
*    load_msg_blakbord_if () -- This routine requests
*    messages on the black board used in the editor. It
*    uses the CLIPS line sent to "ask".
*
*         format : (ask blackboard)
*******************************************************/

#include <process.h>
#include <systids.h>
#include "messages.h"

/*
** EXTERNAL CLIPS FUNCTIONS REFERENCED IN USER
**   FUNCTIONS
*/
     extern struct values *arg_type_check
                              (char *fun_name,
                               int arg_num,
                               char * exp_type);
     extern struct fact   *assert(char * str);

/*******************************************************/

int       load_msg_blakbord_if ()

{
     extern    int                  blakbord_gi;
     extern    assert_mat           assert_gma;

     int                  return_value_i = 0;
     int                  send_results_i;

     assert_gma.struct_id_c = ID_REQUEST_MESSAGE;
     send_results_i =
          send (blakbord_gi,&assert_gma,&assert_gma,
          sizeof(assert_mat),sizeof(assert_mat));
     if (send_results_i != blakbord_gi)
     {
          return_value_i = 1;
     }
```

(mssgblbo.c continued)

```
        else if (assert_gma.struct_id_c ==
             ID_EMPTY_QUEUE)
        {
             sleep (3);
        } else {
             assert (assert_gma.assert_mssg_ca);
        }
        return (return_value_i);
}
```

```
/*****************************************************
*
*                          MSSGENVE.C
*
*****************************************************
*
* load_msg_envelope_if () -- This sends commands and
*    questions to the envelope expert. It takes
*    messages sent through the "ask routine and
*    structures a command to the envelope expert task.
*
*    It would be more object-oriented to include this
*    section in the expert's code, however, it is more
*    efficient to detect errors at the place they
*    occur. This could be thought of as a device driver
*    from the standpoint of the clips environment where
*    the expert is the device and this is its prime
*    interface.
*
*    format : (ask envelope <command>)
*
*    returns :
*         0 -- successful
*         1 -- command is not a word
*         2 -- invalid parameter for command
*         3 -- invalid command
*         4 -- envelope expert mailbox does not exist
*
*****************************************************/

#include <stdio.h>
#include <string.h>
#include <process.h>
#include <systids.h>
#include "clips.h"
#include "constdef.h"
#include "messages.h"

/*
** EXTERNAL CLIPS FUNCTIONS REFERENCED IN USER
**   FUNCTIONS
*/
     extern struct values *arg_type_check
                    (char * fun_name,
                     int arg_num,
                     char * exp_type);
     extern struct fact *assert(char * str);
```

(mssgenve.c continued)

```
/*******************************************************/
int       load_msg_exp_env_if ()

{
     extern    int             exp_env_gi;
     extern    assert_mat      assert_gma;

     char                      command_ca[256];
     int                       send_results_i;
     set_range_mat             *set_range_map;
     set_take_mat              *set_take_map;
     set_low_frequency_mat     *set_low_frequency_map;
     evaluate_edit_mat         *evaluate_edit_map;
     id_only_mat               *id_only_map;
     struct    values          *arg_ptr1,*arg_ptr2;

/*
** Get command from the call from CLIPS
*/
     if ((arg_ptr1 =
          arg_type_check("ask",2,WORD)) == NULL)
     {
          return (1);
     }

/*
** Load messages.
*/
     strcpy (command_ca,arg_ptr1->value);

     /*
     ** Load "sampling rate command"
     */
     if (0 == strcmp (command_ca,"sampling-rate"))
     {
          id_only_map = (id_only_mat *) &assert_gma;
          id_only_map->struct_id_c =
               ID_GET_SAMPLING_RATE;
     }

     /*
     ** Load "set range" command.
     */
     else if (0 == strcmp (command_ca,"set-range"))
     {
          set_range_map = (set_range_mat *)
               &assert_gma;
          set_range_map->struct_id_c = ID_SET_RANGE;
```

(mssgenve.c continued)

```
        if ((arg_ptr1 =
             arg_type_check ("ask",3,NUMBER)) ==
             NULL)
        {
             assert ("error
                  invalid-number-for-range-start
                  envelope-analyzer");
             return (2);
        } else {
             set_range_map->start_of_range_l =
                  (long) arg_ptr1->ivalue;
             if ((arg_ptr2 =
                  arg_type_check
                  ("ask",4,NUMBER)) == NULL)
             {
                  assert ("error
                       invalid-number-for-range-end
                       envelope-analyzer");
                  return (2);
             } else {
                  set_range_map->end_of_range_l =
                       (long) arg_ptr2->ivalue;
             }
        }
}

/*
** Load "evaluate" command.
*/
else if (0 == strcmp (command_ca,"evaluate"))
{
        evaluate_edit_map =
             (evaluate_edit_mat *) &assert_gma;
        evaluate_edit_map->struct_id_c =
             ID_EVALUATE_EDIT;
        if ((arg_ptr1 =
             arg_type_check ("ask",3,WORD)) == NULL)
        {
             assert ("error
                  invalid-take-name-for-evaluation
                  envelope-analyzer");
             return (2);
        } else {
             strcpy (evaluate_edit_map->take_name_ca,
                  arg_ptr1->value);
```

(mssgenve.c continued)

```
            if ((arg_ptr2 =
                 arg_type_check
                 ("ask",4,NUMBER)) == NULL)
            {
                 assert ("error
                      invalid-number-for-start-point
                      envelope-analyzer");
                 return (2);
            }
            evaluate_edit_map->start_of_range_1 =
                 (long) arg_ptr2->ivalue;
            if ((arg_ptr2 =
                 arg_type_check
                 ("ask",5,NUMBER)) == NULL)
            {
                 assert ("error
                      invalid-number-for-end-point
                      envelope-analyzer");
                 return (2);
            }
            evaluate_edit_map->end_of_range_1 =
                 (long) arg_ptr2->ivalue;
       }
}

/*
** Load "set take" command
*/
else if (0 == strcmp (command_ca,"set-take"))
{
       if ((arg_ptr1 =
            arg_type_check ("ask",3,WORD)) == NULL)
       {
            assert ("error invalid-name-for-
                 take-file envelope-analyzer");
            return (2);
       }
       set_take_map = (set_take_mat *) &assert_gma;
       set_take_map->struct_id_c = ID_SET_TAKE;
       strcpy (set_take_map->take_name_ca,
            arg_ptr1->value);
}
```

(mssgenve.c continued)

```
/*
** Load "set take number" command
*/
else if (0 == strcmp (command_ca,
                      "set-take-number"))
{
     if ((arg_ptr1 =
          arg_type_check ("ask",3,WORD)) == NULL)
     {
          assert ("error
               invalid-name-for-take-number
               envelope-analyzer");
          return (2);
     }
     set_take_map = (set_take_mat *) &assert_gma;
     set_take_map->struct_id_c =
          ID_SET_TAKE_NUMBER;
     strcpy (set_take_map->take_name_ca,
          arg_ptr1->value);
}
/*
** Load analyze command.
*/
else if (0 == strcmp (command_ca,"analyze"))
{
     id_only_map = (id_only_mat *) &assert_gma;
     id_only_map -> struct_id_c = ID_ANALYZE;
}
/*
** Load set low frequency command.
*/
else if (0 == strcmp (command_ca,
     "set-low-frequency"))
{
     set_low_frequency_map =
          (set_low_frequency_mat *) &assert_gma;
     set_low_frequency_map -> struct_id_c =
          ID_SET_LOW_FREQUENCY;
     if ((arg_ptr1 =
          arg_type_check ("ask",3,NUMBER)) ==
          NULL)
     {
          assert ("error
               invalid-value-for-low-frequency
               envelope-analyzer");
          return (2);
     }
     set_low_frequency_map -> low_frequency_i =
          arg_ptr1 -> ivalue;
}
```

(mssgenve.c continued)

```
     /*
     ** Load find edit command.
     */
     else if (0 == strcmp (command_ca,"find-edit"))
     {
          id_only_map = (id_only_mat *) &assert_gma;
          id_only_map -> struct_id_c = ID_FIND_EDIT;
     }
     /*
     ** Print an error if message was not legal.
     */
     else
     {
          assert ("error illegal-command
               envelope-analyzer");
          return (3);
     }

/*
** Send message containing command and appropriate
**   parameters to envelope expert task.
*/
     send_results_i =
          send (exp_env_gi,&assert_gma,&assert_gma,
          sizeof(assert_mat),sizeof(assert_mat));

/*
** Check return from expert to see if there was a
**   problem.
*/
     if (send_results_i != exp_env_gi)
     {
          assert ("error sending-message-to-envelope
               decision-maker");
          return (4);
     }

     return (0);
}
```

```
/*********************************************************
*
*                          MSSGRHYT.C
*
*********************************************************
*
* load_msg_rhythm_if () -- This sends commands and
*    questions to the rhythm expert. It takes messages
*    sent through the "ask routine and structures a
*    command to the rhythm expert task.
*
*    It would be more object-oriented to include this
*    section in the expert's code, however, it is more
*    efficient to detect errors at the place they
*    occur. This could be thought of as a device driver
*    from the standpoint of the clips environment where
*    the expert is the device and this is its prime
*    interface.
*
*    format : (ask exp_rhy <command> <parameters> ...)
*
*    returns :
*              0 -- successful
*              1 -- command is not a word
*              2 -- invalid parameter for command
*              3 -- invalid command
*              4 -- rhythm expert mailbox does not
*                   exist
*
*********************************************************/

#include <stdio.h>
#include <string.h>
#include <process.h>
#include <systids.h>
#include "clips.h"
#include "constdef.h"
#include "messages.h"

/*
** EXTERNAL CLIPS FUNCTIONS REFERENCED IN USER
**   FUNCTIONS
*/
     extern struct values *arg_type_check
          (char * fun_name,
           int arg_num,
           char * exp_type);
     extern struct fact   *assert(char * str);
```

(mssgrhyt.c continued)

```
/*******************************************************/

int       load_msg_exp_rhy_if ()

{
     extern    int             exp_rhy_gi;
     extern    assert_mat      assert_gma;

     char                      command_ca[256];
     int                       send_results_i;
     set_range_mat             *set_range_map;
     set_take_mat              *set_take_map;
     id_only_mat               *id_only_map;
     evaluate_edit_mat         *evaluate_edit_map;
     set_edit_to_mat           *set_edit_to_map;
     struct    values          *arg_ptr1,*arg_ptr2;

/*
** Get command from the call from CLIPS
*/
     if ((arg_ptr1 =
          arg_type_check("ask",2,WORD)) == NULL)
     {
          return (1);
     }

/*
** Load messages.
*/
     strcpy (command_ca,arg_ptr1->value);

     /*
     ** Load "set range" command.
     */
     if (0 == strcmp (command_ca,"set-range"))
     {
          set_range_map = (set_range_mat *)
               &assert_gma;
          set_range_map->struct_id_c = ID_SET_RANGE;
          if ((arg_ptr1 =
               arg_type_check ("ask",3,NUMBER)) ==
               NULL)
          {
               assert ("error
                    invalid-number-for-range-start
                    rhythm-analyzer");
               return (2);
          } else {
               set_range_map->start_of_range_1 =
                    (long) arg_ptr1->ivalue;
```

(mssgrhyt.c continued)

```
            if ((arg_ptr2 =
                 arg_type_check
                 ("ask",4,NUMBER)) == NULL)
            {
                 assert ("error
                      invalid-number-for-range-end
                      rhythm-analyzer");
                 return (2);
            } else {
                 set_range_map->end_of_range_1 =
                      (long) arg_ptr2->ivalue;
            }
       }
}

/*
** Load "set take" command
*/
else if (0 == strcmp (command_ca,
     "set-take-number"))
{
     if ((arg_ptr1 =
          arg_type_check ("ask",3,WORD)) == NULL)
     {
          assert ("error invalid-name-for-
               take-file rhythm-analyzer");
          return (2);
     }
     set_take_map = (set_take_mat *) &assert_gma;
     set_take_map->struct_id_c =
          ID_SET_TAKE_NUMBER;
     strcpy (set_take_map->take_name_ca,
          arg_ptr1->value);
}

/*
** Load "set edit to" command
*/
else if (0 == strcmp (command_ca,"set-edit-to"))
{
     if ((arg_ptr1 =
          arg_type_check ("ask",3,NUMBER)) ==
          NULL)
     {
          assert ("error
               invalid-value-for-set-edit-to
               rhythm-analyzer");
          return (2);
     }
```

(mssgrhyt.c continued)

```
     set_edit_to_map =
          (set_edit_to_mat *) &assert_gma;
     set_edit_to_map->struct_id_c =
          ID_SET_EDIT_TO;
     set_edit_to_map->point_1 =
          (long) arg_ptr1->ivalue;
}

/*
** Load "find edit" command
*/
else if (0 == strcmp (command_ca,"find-edit"))
{
     id_only_map = (id_only_mat *) &assert_gma;
     id_only_map -> struct_id_c = ID_FIND_EDIT;
}

/*
** Load "evaluate edit" command
*/
else if (0 == strcmp (command_ca,"evaluate"))
{
     evaluate_edit_map =
          (evaluate_edit_mat *) &assert_gma;
     evaluate_edit_map -> struct_id_c =
          ID_EVALUATE_EDIT;
     if ((arg_ptr1 =
          arg_type_check ("ask",3,WORD)) == NULL)
     {
          assert ("error invalid-name-for-edit
               rhythm-analyzer");
          return (2);
     }
     strcpy (evaluate_edit_map -> take_name_ca,
          arg_ptr1->value);
     evaluate_edit_map->start_of_range_1 =
          (long) arg_ptr1->ivalue;
     if ((arg_ptr2 =
          arg_type_check ("ask",4,NUMBER)) ==
          NULL)
     {
          assert ("error
               invalid-number-for-edit-start
               rhythm-analyzer");
          return (2);
     }
     evaluate_edit_map->start_of_range_1 =
          (long) arg_ptr2->ivalue;
```

(mssgrhyt.c continued)

```
            if ((arg_ptr2 =
                 arg_type_check ("ask",5,NUMBER)) ==
                 NULL)
            {
                 assert ("error invalid-number-for-
                      edit-end rhythm-analyzer");
                 return (2);
            }
            evaluate_edit_map->end_of_range_1 =
                 (long) arg_ptr2->ivalue;
       }

       /*
       ** Print an error if message was not legal.
       */
       else
       {
            assert ("error illegal-command
                 rhythm-analyzer");
            return (3);
       }

/*
** Send message containing command and appropriate
**   parameters to spectrum expert task.
*/
       send_results_i =
            send (exp_rhy_gi,&assert_gma,&assert_gma,
            sizeof(assert_mat),sizeof(assert_mat));

/*
** Check return from expert to see if there was a
**   problem.
*/
       if (send_results_i != exp_rhy_gi)
       {
            assert ("error
                 sending-message-to-rhythm-analyzer
                 decision-maker");
            return (4);
       }

       return (0);
}
```

```
/*****************************************************
*
*                          MSSGSPEC.C
*
*****************************************************
*
* load_msg_spectrum_if () -- This sends commands and
*    questions to the spectrum expert. It takes
*    messages sent through the "ask routine and
*    structures a command to the spectrum expert task.
*
*    It would be more object-oriented to include this
*    section in the expert's code, however, it is more
*    efficient to detect errors at the place they
*    occur. This could be thought of as a device driver
*    from the standpoint of the clips environment where
*    the expert is the device and this is its prime
*    interface.
*
*    format : (ask spectrum <command>)
*
*    returns :
*         0 -- successful
*         1 -- command is not a word
*         2 -- invalid parameter for command
*         3 -- invalid command
*         4 -- spectrum expert mailbox does not exist
*
*****************************************************/

#include <stdio.h>
#include <string.h>
#include <process.h>
#include <systids.h>
#include "clips.h"
#include "constdef.h"
#include "messages.h"

/*
** EXTERNAL CLIPS FUNCTIONS REFERENCED IN USER
**   FUNCTIONS
*/
     extern struct values *arg_type_check
               (char * fun_name,
                int arg_num,
                char * exp_type);
     extern struct fact   *assert(char * str);
```

(mssgspec.c continued)

```
/*****************************************************/

int       load_msg_exp_fft_if ()

{
     extern    int            exp_fft_gi;
     extern    assert_mat     assert_gma;

     char                     command_ca[256];
     int                      send_results_i;
     set_range_mat            *set_range_map;
     set_take_mat             *set_take_map;
     id_only_mat              *id_only_map;
     sample_rate_mat          *sample_rate_map;
     evaluate_edit_mat        *evaluate_edit_map;
     struct    values         *arg_ptr1,*arg_ptr2;

/*
** Get command from the call from CLIPS
*/
     if ((arg_ptr1 =
          arg_type_check("ask",2,WORD)) == NULL)
     {
          return (1);
     }
/*
** Load messages.
*/
     strcpy (command_ca,arg_ptr1->value);

     /*
     ** Load "sampling rate command"
     */
     if (0 == strcmp (command_ca,"set-sample-rate"))
     {
          sample_rate_map =
               (sample_rate_mat *) &assert_gma;
          sample_rate_map->struct_id_c =
               ID_SET_SAMPLING_RATE;
          if ((arg_ptr1 =
               arg_type_check ("ask",3,NUMBER)) ==
               NULL)
          {
               assert ("error
                    invalid-number-for-sampling-rate
                    spectrum-analyzer");
               return (2);
          }
```

(mssgspec.c continued)

```
     sample_rate_map->rate_l =
          (long) arg_ptr1->ivalue;
}

/*
** Load "set range" command.
*/
else if (0 == strcmp (command_ca,"set-range"))
{
     set_range_map = (set_range_mat *)
          &assert_gma;
     set_range_map->struct_id_c = ID_SET_RANGE;
     if ((arg_ptr1 =
          arg_type_check ("ask",3,NUMBER)) ==
          NULL)
     {
          assert ("error
               invalid-number-for-range-start
               spectrum-analyzer");
          return (2);
     } else {
          set_range_map->start_of_range_l =
               (long) arg_ptr1->ivalue;
          if ((arg_ptr2 =
               arg_type_check
               ("ask",4,NUMBER)) == NULL)
          {
               assert ("error
                    invalid-number-for-range-end
                    spectrum-analyzer");
               return (2);
          } else {
               set_range_map->end_of_range_l =
                    (long) arg_ptr2->ivalue;
          }
     }
}

/*
** Load "evaluate" command.
*/
else if (0 == strcmp (command_ca,"evaluate"))
{
     evaluate_edit_map =
          (evaluate_edit_mat *) &assert_gma;
     evaluate_edit_map->struct_id_c =
          ID_EVALUATE_EDIT;
```

(mssgspec.c continued)

```
     if ((arg_ptr1 =
          arg_type_check ("ask",3,WORD)) == NULL)
     {
          assert ("error
               invalid-name-for-output-file
               spectrum-analyzer");
          return (2);
     }
     strcpy (evaluate_edit_map->take_name_ca,
          arg_ptr1->value);
     if ((arg_ptr2 =
          arg_type_check ("ask",4,NUMBER)) ==
          NULL)
     {
          assert ("error
               invalid-number-for-range-start
               spectrum-analyzer");
          return (2);
     }
     evaluate_edit_map->start_of_range_1 =
          (long) arg_ptr2->ivalue;
     if ((arg_ptr2 =
          arg_type_check ("ask",5,NUMBER)) ==
          NULL)
     {
          assert ("error
               invalid-number-for-range-end
               spectrum-analyzer");
          return (2);
     }
     evaluate_edit_map->end_of_range_1 =
          (long) arg_ptr2 -> ivalue;
}
/*
** Load "set take" command
*/
else if (0 == strcmp (command_ca,"set-take"))
{
     if ((arg_ptr1 =
          arg_type_check ("ask",3,WORD)) == NULL)
     {
          assert ("error invalid-name-for-
               take-file spectrum-analyzer");
          return (2);
     }
     set_take_map = (set_take_mat *) &assert_gma;
     set_take_map->struct_id_c = ID_SET_TAKE;
     strcpy (set_take_map->take_name_ca,
          arg_ptr1->value);
}
```

(mssgspec.c continued)

```
     /*
     ** Load "find edit" command
     */
     else if (0 == strcmp (command_ca,"find-edit"))
     {
          id_only_map = (id_only_mat *) &assert_gma;
          id_only_map -> struct_id_c = ID_FIND_EDIT;
     }
     /*
     ** Load "get low frequency" command
     */
     else if (0 == strcmp
          (command_ca,"get-low-frequency"))
     {
          id_only_map = (id_only_mat *) &assert_gma;
          id_only_map -> struct_id_c =
               ID_GET_LOW_FREQUENCY;
     }

     /*
     ** Print an error if message was not legal.
     */
     else
     {
          assert ("error illegal-command
               spectrum-analyzer");
          return (3);
     }

/*
** Send message containing command and appropriate
**   parameters to spectrum expert task.
*/
     send_results_i =
          send (exp_fft_gi,&assert_gma,&assert_gma,
          sizeof(assert_mat),sizeof(assert_mat));

/*
** Check return from expert to see if there was a
**   problem.
*/
     if (send_results_i != exp_fft_gi)
     {
          assert ("error sending-message-to-spectrum
               decision-maker");
          return (4);
     }

     return (0);
}
```

```
/*********************************************************
*
*                          MAKKEEDIT.C
*
*********************************************************
*
* It is the purpose of this program to construct an
*    output file from the edit points and source files.
*    It calls the functions adjust_header_vf and
*    crossfade_vf to assist in generating the output
*    file. It does not necessarily mean the output of
*    this process is a usable edit. It only generates
*    attempts to be tested.
*
*********************************************************/

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "constdef.h"
#include "structde.h"
#include "messages.h"
#include "globals.h"
#include "protos.h"
#include "userdefs.h"

int       make_edit ()

{
     extern int arg_num_check(char * fun_name,
                              int check_val,
                              int exp_num);
     extern struct values *arg_type_check
          (char * fun_name,
           int arg_num,
           char * exp_type);
     extern struct fact   *assert(char * str);

     extern    char      directory_gca[];
     extern    FILE      *text_out_gfh;

               char      destination_ca[17];
               char      take1_ca[17];
               char      take2_ca[17];
               char      file_name1_ca[256];
               char      file_name2_ca[256];
               char      destination_name_ca[256];
               int       *buffer1_ip;
               int       *buffer2_ip;
               int       buffer2_length_i;
               int       *dest_buffer_ip;
               int       crossfade_length_i;
```

(makeedit.c continued)

```
                long        sampling_rate_l;
                long        take1_edit_l;
                long        take2_edit_l;
                long        read_position1_l;
                long        read_position2_l;
                FILE        *take1_fhp;
                FILE        *take2_fhp;
                FILE        *dest_fhp;

        struct    values    *arg_ptr;


        fprintf (text_out_gfh,
             "\decision maker creating edited file.");

/*
** Check to see if argument count is correct. Arguments
**   are:
**        target file name                -- string
**        source 1 file name              -- string
**        source 1 edit point             -- float
**        source 2 file name              -- string
**        source 2 edit point             -- float
**        sampling rate                   -- float
*/
        if (arg_num_check ("make-edit",EXACTLY,6) == -1)
        {
             return (0.0);
        }

/*
** Get destination file name. Check that it is a
**   string.
*/
        if ((arg_ptr =
             arg_type_check ("make-edit",1,WORD)) == NULL)

        {
             return (0.0);
        }
        strcpy (destination_ca,arg_ptr->value);
```

(makeedit.c continued)

```
/*
** Get the take name for the first take.
*/
     if ((arg_ptr =
          arg_type_check ("make-edit",2,WORD)) == NULL)

     {
          return (0.0);
     }
     strcpy (take1_ca,arg_ptr->value);
/*
** Get the point in the file at which to make the edit
**   for the outgoing sequence.
*/
     if ((arg_ptr =
          arg_type_check ("make-edit",3,NUMBER)) ==
          NULL)
     {
          return (0.0);
     }
     take1_edit_1 = arg_ptr->ivalue;

/*
** Get the take name for the second take.
*/
     if ((arg_ptr =
          arg_type_check ("make-edit",4,WORD)) == NULL)
     {
          return (0.0);
     }
     strcpy (take2_ca,arg_ptr->value);

/*
** Get the point in the file at which to make the edit
**   for the incoming sequence.
*/
     if ((arg_ptr =
          arg_type_check ("make-edit",5,NUMBER)) ==
          NULL)
     {
          return (0.0);
     }
     take2_edit_1 = arg_ptr->ivalue;
```

(makeedit.c continued)

```
/*
** Get the sampling rate for the two sequences.
*/
     if ((arg_ptr =
          arg_type_check ("make-edit",6,NUMBER)) ==
          NULL)
     {
          return (0.0);
     }
     sampling_rate_1 = arg_ptr->ivalue;

/*
** Get some memory to do the transfer in.
*/
     buffer1_ip = (int *) calloc
          (XFER_BUFFER_SIZE,sizeof (int));
     buffer2_ip = (int *) calloc
          (XFER_BUFFER_SIZE,sizeof (int));
     dest_buffer_ip = (int *) calloc (XFER_BUFFER_SIZE,
          sizeof (int));

/*
** Open the source file for the outgoing take for
**   reading.
*/
     strcpy (file_name1_ca,directory_gca);
     strcat (file_name1_ca,take1_ca);
     take1_fhp = fopen (file_name1_ca,"r");

/*
** Open the source file for the incoming take for
**   reading.
*/
     strcpy (file_name2_ca,directory_gca);
     strcat (file_name2_ca,take2_ca);
     take2_fhp = fopen (file_name2_ca,"r");

/*
** Open the destination file for the completed edit for
**   writing.
*/
     strcpy (destination_name_ca,directory_gca);
     strcat (destination_name_ca,destination_ca);
     dest_fhp = fopen (destination_name_ca,"w");
```

(makeedit.c continued)

```
/*
** Adjust header information for output file.
*/
     adjust_headers_vf (&take1_fhp, &take2_fhp,
                        &dest_fhp, take1_edit_l,
                        take2_edit_l);
     read_position1_l = HEADER_SIZE;
     read_position2_l = HEADER_SIZE;

/*
** Copy the first part of the source file to the
**   destination file.
*/
     while ((read_position1_l + XFER_BUFFER_SIZE) <
             (take1_edit_l - (sampling_rate_l / 100)))
     {
          fread ((void *) buffer1_ip, sizeof (int),
               XFER_BUFFER_SIZE, take1_fhp);
          fwrite ((void *) buffer1_ip, sizeof (int),
               XFER_BUFFER_SIZE, dest_fhp);
          read_position1_l += XFER_BUFFER_SIZE;
     }
/*
** Copy remaining section up to crossfade region.
*/
     crossfade_length_i = (int) (sampling_rate_l /
          100);
     fread ((void *) buffer1_ip, sizeof (int),
          (int) (take1_edit_l -
          (long) crossfade_length_i -
          read_position1_l), take1_fhp);
     fwrite ((void *) buffer1_ip, sizeof (int),
          (int) (take1_edit_l -
          (long) crossfade_length_i -
          read_position1_l), dest_fhp);

/*
** Load crossfade buffers.
*/
     fread ((void *) buffer1_ip, sizeof (int),
          crossfade_length_i, take1_fhp);
     fseek (take2_fhp,((take2_edit_l + HEADER_SIZE) *
          sizeof (int)),SEEK_SET);
     fread ((void *) buffer2_ip, sizeof (int),
          crossfade_length_i, take2_fhp);
```

(makeedit.c continued)

```
/*
** Generate the crossfade.
*/
     make_crossfade_vf (buffer1_ip, buffer2_ip,
          dest_buffer_ip, crossfade_length_i);
     fwrite ((void *) dest_buffer_ip, sizeof (int),
          crossfade_length_i, dest_fhp);

/*
** Copy the remainder of the second file to destination
**   to finish up.
*/
     while (buffer2_length_i =
          fread ((void *) buffer2_ip, sizeof (int),
               XFER_BUFFER_SIZE, take2_fhp))
     {
          fwrite ((void *) buffer2_ip, sizeof (int),
               buffer2_length_i, dest_fhp);
     }

/*
** Close all affected files, free the memory, and exit.
*/
     free (buffer1_ip);
     free (buffer2_ip);
     free (dest_buffer_ip);
     fclose (take1_fhp);
     fclose (take2_fhp);
     fclose (dest_fhp);
     fprintf (text_out_gfh,"\decision maker completed
          edit attempt.");
     return (0);
}
```

```
/*********************************************************
*
*                          ADJUSTHDR.C
*
*********************************************************
*
* It is the purpose of this program to construct an
*    output header with information appropriate for the
*    HyperSignal software, so that it can be played
*    back.
*
*********************************************************/

#include <stdio.h>
#include "userdefs.h"

void adjust_headers_vf (FILE **take1_fhp,
                        FILE **take2_fhp,
                        FILE **dest_fhp,
                        long take1_edit_l,
                        long take2_edit_l)

{
     extern    FILE *text_out_gfh;

     int       header1_ia[HEADER_SIZE],
               header2_ia[HEADER_SIZE],
               dest_header_ia[HEADER_SIZE];

     long dest_num_data_l;
     long num_data_take2_l;



     fprintf (text_out_gfh,
          "\decision maker adjusting header
          information.");

/*
** get headers from existing source files.
*/
     fread ((void *) header1_ia, sizeof (int),
          HEADER_SIZE,*take1_fhp);
     fread ((void *) header2_ia, sizeof (int),
          HEADER_SIZE,*take2_fhp);
```

(adjusthdr.c continued)

```
/*
** Take whichever amplitude is larger.
*/
     if (header1_ia[AMPLITUDE] > header2_ia[AMPLITUDE])
     {
          dest_header_ia[AMPLITUDE] =
               header1_ia[AMPLITUDE];
     } else {
          dest_header_ia[AMPLITUDE] =
               header2_ia[AMPLITUDE];
     }

/*
** It is assumed for this research that the frame sizes
**   are equal.
*/
     dest_header_ia[FRAMESIZE] = header1_ia[FRAMESIZE];


/*
** To make the edit, both sampling frequencies are
**   guaranteed to be the same.
*/
     dest_header_ia[SAMPLING_FREQ_R] =
          header1_ia[SAMPLING_FREQ_R];
     dest_header_ia[SAMPLING_FREQ_D] =
          header1_ia[SAMPLING_FREQ_D];

/*
** FFT order is not used for these experiments.
*/
     dest_header_ia[FFT_ORDER] = header1_ia[FFT_ORDER];

/*
** Frame overlap, data type, and user defined areas are
**   not used in this research.
*/
     dest_header_ia[FRAME_OVERLAP] =
          header1_ia[FRAME_OVERLAP];
     dest_header_ia[DATA_TYPE] = header1_ia[DATA_TYPE];

     dest_header_ia[USER1] = header1_ia[USER1];

/*
** To determine the number of data elements used by the
**   outgoing source, take the current edit position
**   and subtract the header length.
*/
     dest_num_data_l = take1_edit_l - HEADER_SIZE;
```

(adjusthdr.c continued)

```
/*
** Determine the number of data points in the second
**   file.
*/
     num_data_take2_l =
          (long) ((long) header2_ia[NUM_DATA_D] *
          (long) 16384 + (long) header2_ia[NUM_DATA_R]
          + (long) HEADER_SIZE);
     num_data_take2_l -= take2_edit_l;

/*
** Determine the number of data points in the new file.
*/
     dest_num_data_l += num_data_take2_l;

/*
** Load header with new data length.
*/
     dest_header_ia[NUM_DATA_D] =
          (int) (dest_num_data_l / 16384);
     dest_header_ia[NUM_DATA_R] =
          (int) (dest_num_data_l % 16384);

/*
** Write header to output file.
*/
     fwrite ((void *) dest_header_ia, sizeof (int),
          HEADER_SIZE, *dest_fhp);
     return;
}
```

```
/*******************************************************
*
*                          CROSSFAD.C
*
*******************************************************
*
* This program takes the two input buffers and
*    generates an amplitude decrease in buffer one
*    while producing an equivalent amplitude increase
*    in buffer two. This causes the two input files to
*    switch gradually (crossfade).
*
*******************************************************/

#include <stdio.h>

void make_crossfade_vf (int buffer1_ip[],
                        int buffer2_ip[],
                        int dest_buffer_ip[],
                        int crossfade_length_i)

{
     extern    FILE *text_out_gfh;

     int       index_i;
     float     mult1_f,mult2_f,factor_f;

     fprintf (text_out_gfh,
          "\decision maker is generating a
          crossfade.");
/*
** Determine a smooth cross fade between incoming and
**   outgoing files. This method does not account for
**   logarithmic effect of amplitude.
*/
     factor_f = 1.0 / (float) crossfade_length_i;
     mult1_f = 1.0;
     mult2_f = 0.0;
     for (index_i = 0; index_i < crossfade_length_i;
           index_i++, mult1_f -= factor_f,
           mult2_f += factor_f)
     {
          buffer1_ip[index_i] =
               (int) ((float) buffer1_ip[index_i] *
               mult1_f);
          buffer2_ip[index_i] =
               (int) ((float) buffer2_ip[index_i] *
               mult2_f);
          dest_buffer_ip[index_i] =
               buffer1_ip[index_i] +
               buffer2_ip[index_i];
     }
```

(crossfad.c continued)

```
    return;
}
```

```
;
; ----------- AI DIGITAL AUDIO EDITOR ----------------
;                    MAIN RULES
;
;*****************************************************
;
; This section sets up the experts and their mailboxes
;    (agents).
;*****************************************************
;
; Initialization.
;

(defrule start-experts ""
     ?start-fact <- (initial-fact)
=>
     (retract ?start-fact)
     (initialize-experts)
     (assert (check-messages))
     (assert (goal plan-edit))
     (assert (goal execute-edit))
     (assert (goal learn-from-edit)))

;
; If there is nothing else to do, the program checks
;    for messages to determine if any information has
;    been returned by an expert.
(defrule check-messages ""
     (declare (salience -10))
     ?chk-mssg-fact <- (check-messages)
=>
     (retract ?chk-mssg-fact)
     (ask blackboard)
     (assert (check-messages)))
```

(rules continued)

```
;
; This rule handles program termination if one of the
;    experts signals a low level error. It preempts all
;    normal processing and terminates after printing a
;    message.

(defrule error-handler ""
     (declare (salience 10))
     (error ?type ?by-who)
=>
     (printout "Error " ?type " was registered by "
          ?by-who)
     (exit))

;*****************************************************
; This section is used for planning an edit.
;    Information must be obtained from the user about
;    the source files for the two audio streams which
;    will be cut and linked together for the edit.
;*****************************************************
;
; if the goal is to plan an edit, we need :
;         2 source files.
;         a destination file.
;         a sample rate for each of the files.
;         a range of samples within the files that
;              represents the crest of the note which
;              will be the note before the edit to the
;              crest of the note which will be after
;              the edit.
(defrule get-source-info ""
     (goal plan-edit)
=>
     (assert (goal get second sampling-frequency))
     (assert (goal get first sampling-frequency))
     (assert (goal get second range))
     (assert (goal get first range))
     (assert (goal get output take))
     (assert (goal get second take))
     (assert (goal get first take)))
```

(rules continued)

```
;
; Ask user for information. This section could be
;    replaced with a graphic section.
(defrule get-take ""
     ?old-goal <- (goal get ?number take)
=>
     (printout "What is the name of the file for the "
          ?number " take?  ")
     (bind ?answer (read))
     (assert (?number take-is ?answer))
     (retract ?old-goal))

(defrule get-sampling-frequency ""
     ?old-goal <- (goal get ?number sampling-frequency)
     (?number take-is ?name)
=>
     (ask envelope set-take-number ?number)
     (ask envelope set-take ?name)
     (ask envelope sampling-rate)
     (retract ?old-goal))

(defrule same-take-sampling-rates ""
     (first take-is ?name)
     (second take-is ?name)
     ?have-goal <- (goal get second sampling-frequency)
=>
     (retract ?have-goal)
     (assert (have sampling-rates)))

(defrule different-take-sampling-rates ""
     (first take-is ?name1)
     (second take-is ?name2&~?name1)
     (?name1 sampling-rate-is ?rate)
     (?name2 sampling-rate-is ?rate)
=>
     (assert (have sampling-rates)))
```

(rules continued)

```
(defrule sample-rate-error ""
     ?old-first-take <- (first take-is ?name1)
     ?old-second-take <- (second take-is
          ?name2&~?name1)
     ?old-rate1 <- (?name1 sampling-rate-is ?rate1)
     ?old-rate2 <- (?name2 sampling-rate-is
          ?rate2&~?rate1)
=>
     (printout "Sampling rates do not match" crlf)
     (retract ?old-first-take)
     (retract ?old-second-take)
     (retract ?old-rate1)
     (retract ?old-rate2)
     (assert (goal get second take))
     (assert (goal get first take)))

(defrule change-fft-sampling-rate ""
     (have sampling-rates)
     (?name sampling-rate-is ?rate)
     ?f1 <- (spectrum sampling-rate-is ?rate2&~?rate)
=>
     (retract ?f1)
     (ask spectrum set-sample-rate ?rate))

(defrule set-fft-sampling-rate ""
     (have sampling-rates)
     (?name sampling-rate-is ?rate)
     (not (spectrum sampling-rate-is ?rate))
=>
     (ask spectrum set-sample-rate ?rate))

(defrule get-range ""
     ?old-goal <- (goal get ?number range)
=>
     (printout
          "What is the start of the last note before
          the edit " crlf " from the " ?number " take?
          ")
     (bind ?start (read))
     (printout "What is the start of the first note
          after the edit " crlf " from the " ?number "
          take? ")
     (bind ?end (read))
     (assert (?number range-is ?start ?end))
     (retract ?old-goal))
```

(rules continued)

```
(defrule range-error ""
     ?old-range <- (?number range-is ?start ?end&:
          (<= ?end ?start))
=>
     (printout "start of note is after end of note"
          crlf)
     (retract ?old-range)
     (assert (goal get ?number range)))

(defrule end-plan-edit ""
     ?end-phase <- (goal plan-edit)
     (or (have sampling-rates)
          (and (?take1 sampling-rate-is ?rate)
               (?take2 sampling-rate-is ?rate)))
     (output take-is ?outtake)
     (first range-is ?start1 ?end1)
     (second range-is ?start2 ?end2)
=>
     (retract ?end-phase)
     (assert (edit planned)))

;-----------------------------------------------------
; This section is used to execute the edit. It can be
;    thought of as the main control loop.
;-----------------------------------------------------
(defrule do-an-edit ""
     (goal execute-edit)
     (edit planned)
=>
     (assert (goal get edit-point))
     (assert (goal make-edit))
     (assert (goal evaluate edit-point)))

(defrule get-first-edit-point ""
     (goal execute-edit)
     (goal get edit-point)
=>
     (assert (goal get first edit-point)))

(defrule erase-get-edit-point-goal ""
     ?f1 <- (goal get ?take edit-point)
     (?take edit-point-is ?place)
=>
     (retract ?f1))
```

(rules continued)

```
(defrule get-second-edit-point ""
     (goal execute-edit)
     (goal get edit-point)
     (first edit-point-is ?place)
=>
     (assert (goal get second edit-point)))

(defrule remove-get-second-edit-point ""
     ?f1 <- (goal get second edit-point)
     (second edit-point-is ?point)
=>
     (retract ?f1))

(defrule have-edits ""
     ?f1 <- (goal get edit-point)
     ?f2 <- (first edit-point-is ?point1)
     ?f3 <- (second edit-point-is ?point2)
=>
     (retract ?f1)
     (retract ?f2)
     (retract ?f3)
     (assert (edit-points-at ?point1 ?point2)))

(defrule make-edit ""
     ?f1 <- (goal make-edit)
     (output take-is ?output)
     (first take-is ?name1)
     (second take-is ?name2)
     (edit-points-at ?point1 ?point2)
     (?name1 sampling-rate-is ?rate)
=>
     (make-edit ?output ?name1 ?point1 ?name2 ?point2
          ?rate)
     (assert (output edit-is ?point1))
     (retract ?f1))

(defrule evaluate-edit ""
     (goal evaluate edit-point)
     (output take-is ?output)
     (output edit-is ?point)
     (edit-points-at ?point1 ?point2)
=>
     (ask envelope evaluate ?output ?point ?point)
     (ask spectrum evaluate ?output ?point ?point)
     (ask rhythm evaluate ?output ?point1 ?point2))
```

(rules continued)

```
;-------------------------------------------------------
; This section is used for getting the initial attempt
;    at an edit. This is done before any information
;    has been gathered about the points which should be
;    edited.
;-------------------------------------------------------
(defrule set-defaults ""
     (goal get ?take edit-point)
     (?take take-is ?name)
     (?name sampling-rate-is ?rate)
     (?take range-is ?start ?end)
=>
     (ask envelope set-take-number ?take)
     (ask envelope set-take ?name)
     (assert (set envelope set-low-frequency 50))
     (assert (set envelope set-range ?start ?end))
     (ask spectrum set-take ?name)
     (assert (set spectrum set-range ?start ?end))
     (ask rhythm set-take-number ?take)
     (assert (set rhythm set-range ?start ?end)))

(defrule set-range ""
     ?f1 <- (set ?expert set-range ?start ?stop)
     (not (?expert range-is ?anystart ?anystop))
=>
     (retract ?f1)
     (ask ?expert set-range ?start ?stop))

(defrule change-range ""
     ?f1 <- (set ?expert set-range ?start ?stop)
     ?f2 <- (?expert range-is ?anystart ?anystop)
=>
     (retract ?f1)
     (retract ?f2)
     (ask ?expert set-range ?start ?stop))

(defrule set-low-frequency ""
     ?f1 <- (set envelope set-low-frequency ?value)
     (not (envelope low-frequency-is ?other_value))
=>
     (retract ?f1)
     (ask envelope set-low-frequency ?value))

(defrule change-low-frequency ""
     ?f1 <- (set envelope set-low-frequency ?value)
     ?f2 <- (envelope low-frequency-is ?other_value)
=>
     (retract ?f1)
     (retract ?f2)
     (ask envelope set-low-frequency ?value))
```

(rules continued)

```
(defrule exp-env-analyze ""
     (envelope low-frequency-is ?freq)
     (envelope range-is ?start ?stop)
     (envelope take-is ?name)
     (have sampling-rates)
=>
     (ask envelope analyze))

(defrule get-env-edit ""
     (goal get ?take edit-point)
     (envelope take-is ?name)
     (have sampling-rates)
     (envelope range-is ?start ?stop)
     (?take range-is ?start ?stop)
     (envelope analyzed ?start ?stop)
=>
     (ask envelope find-edit))

(defrule tell-first-edit-to-rhythm ""
     (first take-is ?name)
     (first range-is ?start ?stop)
     (envelope ?name1 ?start ?stop edit-at ?place)
     (not (rhythm ?take ?anystart ?anystop edit-at
          ?anyplace))
=>
     (ask rhythm set-edit-to ?place))

(defrule change-first-edit-to-rhythm ""
     (first take-is ?name)
     (first range-is ?start ?stop)
     ?f1 <- (rhythm first ?start ?stop edit-at ?place)
     (envelope ?name ?start ?stop edit-at
          ?place2&~?place)
=>
     (retract ?f1))

(defrule get-first-rhythm-edit-point ""
     (first edit-point-is ?place)
=>
     (ask rhythm find-edit))

(defrule get-fft-edit ""
     (goal get ?take edit-point)
     (spectrum take-is ?name)
     (have sampling-rates)
     (spectrum range-is ?start ?stop)
     (?take range-is ?start ?stop)
=>
     (ask spectrum find-edit))
```

(rules continued)

```
;-----------------------------------------------------
; This section is used when there is a disagreement
;    between experts. First an attempt is made to
;    gather information from the individual experts and
;    use that information to re-evaluate their choices.
;-----------------------------------------------------
; Because the rhythm expert does not read files, we
;    substitute the take name for the take number for
;    the rhythm expert returns.
(defrule rhythm-expert-adjustment ""
     (?take take-is ?name)
     ?f1 <- (rhythm ?take ?start ?stop edit-at ?place)
=>
     (retract ?f1)
     (assert (rhythm ?name ?start ?stop edit-at
          ?place)))

(defrule minor-deviation ""
     (goal get ?take edit-point)
     (?take take-is ?name)
     (?name sampling-rate-is ?rate)
     (?take range-is ?start ?stop)
     ?f1 <- (?expert1 ?name ?start ?stop edit-at
          ?exp1-place)
     ?f2 <- (?expert2&~?expert1 ?name ?start ?stop
          edit-at ?exp2-place&:
          (<= (abs (- ?exp1-place ?exp2-place))
               (/ ?rate 100)))
     ?f3 <- (?expert3&~?expert1&~?expert2 ?name ?start
          ?stop edit-at ?exp3-place&:
          (and (<= (abs
                    (- ?exp1-place ?exp3-place))
                    (/ ?rate 100))
               (<= (abs (- ?exp2-place ?exp3-place))
                    (/ ?rate 100))))
=>
     (retract ?f1)
     (retract ?f2)
     (retract ?f3)
     (bind ?center-point (/ (+ ?exp1-place ?exp2-place)
          2))
     (printout "Agreement has been reached on edit
          point placement" crlf)
     (assert (?take edit-point-is ?center-point)))
```

(rules continued)

```
(defrule no-spectrum-low-freq ""
     ?f1 <- (envelope low-frequency-is ?freq)
     ?f2 <- (envelope analyzed ?start ?stop)
     (?name sampling-rate-is ?rate)
     ?f3 <- (envelope ?name ?start ?stop edit-at
          ?env-place)
     ?f4 <- (rhythm ?name ?start ?stop edit-at
          ?env-place)
     (spectrum ?name ?start ?stop edit-at ?fft-place&:
          (> (abs (- ?env-place ?fft-place))
               (/ ?rate 100)))
     (not (spectrum low-frequency-is ?freq))
=>
     (retract ?f1)
     (retract ?f2)
     (retract ?f3)
     (retract ?f4)
     (printout "Backtracking because of disagreement.
          Trying for frequency." crlf)
     (ask spectrum get-low-frequency))

(defrule spectrum-low-freq-to-envelope ""
     (spectrum low-frequency-is ?number)
     (not (envelope low-frequency-is ?number))
=>
     (ask envelope set-low-frequency ?number))

(defrule shift-earlier-is-preferred ""
     (?take name-is ?name)
     (?name sampling-rate-is ?rate)
     ?f1 <- (?expert1 ?name ?start ?stop edit-at
          ?exp1-place)
     (?expert2 ?name ?start ?stop edit-at ?exp2-place&:

          (> (abs (- ?exp1-place ?exp2-place))
               (/ ?rate 100)))
     (test (> ?exp1-place ?exp2-place))
     (?expert3 ?name ?start ?stop edit-at ?exp3-place&:

          (<= (abs (- ?exp3-place ?exp2-place))
               (/ ?rate 100)))
=>
     (retract ?f1)
     (assert (?expert1 should-be-earlier ?take)))
```

(rules continued)

```
(defrule shift-later-is-preferred ""
     (?take name-is ?name)
     (?name sampling-rate-is ?rate)
     ?f1 <- (?expert1 ?name ?start ?stop edit-at
          ?exp1-place)
     (?expert2 ?name ?start ?stop edit-at ?exp2-place&:

          (> (abs (- ?exp1-place ?exp2-place))
               (/ ?rate 100)))
     (test (< ?exp1-place ?exp2-place))
     (?expert3 ?name ?start ?stop edit-at ?exp3-place&:

          (<= (abs (- ?exp3-place ?exp2-place))
               (/ ?rate 100)))
=>
     (retract ?f1)
     (assert (?expert1 should-be-later ?take)))

;-----------------------------------------------------
; Rules need to be added to this section to reset the
;    take information for the expert to ask for a later
;    or earlier edit point. It has not been included,
;    because it was not used in the example and is
;    still largely untested.
;-----------------------------------------------------
(defrule abort-on-loop-back-earlier ""
     (declare (salience -5))
     (?expert1 should-be-earlier ?take)
=>
     (assert (goal reject-edit)))

(defrule abort-on-loop-back-later ""
     (declare (salience -5))
     (?expert1 should-be-later ?take)
=>
     (assert (goal reject-edit)))
```

(rules continued)

```
;-----------------------------------------------------
; This section is used to evaluate the edit. To
;    simplify the experiments, the acceptability
;    parameters have been hard-coded into the experts
;    themselves. An acceptability frame section could
;    be added to this to inform the experts as to what
;    is considered acceptable.
;-----------------------------------------------------
(defrule a-ok ""
     ?f1 <- (goal evaluate edit-point)
     ?f2 <- (envelope accepts edit)
     ?f3 <- (spectrum accepts edit)
     ?f4 <- (rhythm accepts edit)
=>
     (retract ?f1)
     (retract ?f2)
     (retract ?f3)
     (retract ?f4)
     (assert (goal accept-edit)))

;-----------------------------------------------------
; This section alters edit based on response from the
;    experts to evaluation. This area requires a good
;    deal of expansion to handle more general cases.
;-----------------------------------------------------
(defrule shift-point-earlier ""
     (?expert1 prefers-earlier ?take)
     (?expert2 prefers-earlier¦accepts ?take¦edit)
     (?expert3 prefers-earlier¦accepts ?take¦edit)
=>
     (assert (?expert2 should-be-earlier ?take))
     (assert (?expert3 should-be-earlier ?take)))

(defrule shift-point-later ""
     (?expert1 prefers-later ?take)
     (?expert2 prefers-later¦accepts ?take¦edit)
     (?expert3 prefers-later¦accepts ?take¦edit)
=>
     (assert (?expert2 should-be-later ?take))
     (assert (?expert3 should-be-later ?take)))
```

(rules continued)

```
;-------------------------------------------------------
; This section accepts edits. It asks the user if he
;    agrees with the edit that was done.
;-------------------------------------------------------
(defrule accept-edit-point
     ?f1 <- (goal execute-edit)
     ?f2 <- (goal accept-edit)
=>
     (retract ?f1)
     (retract ?f2)
     (printout "Do you approve of this edit? (y/n)"
          crlf)
     (bind ?response (read))
     (assert (goal get-another-point-from-user
          ?response)))

(defrule get-another ""
     (goal get-another-point-from-user n)
=>
     (printout "Could you find a better point? (y/n)"
          crlf)
     (bind ?response (read))
     (assert (goal examine-response ?response)))

(defrule chose-another ""
     ?f1 <- (goal get-another-point-from-user n)
     ?f2 <- (goal examine-response y)
=>
     (retract ?f1)
     (retract ?f2)
     (assert (edit-succeeded user-reports another)))

(defrule should-have-failed ""
     ?f1 <- (goal get-another-point-from-user n)
     ?f2 <- (goal examine-response n)
=>
     (retract ?f1)
     (retract ?f2)
     (assert (edit-succeeded user-reports n)))

(defrule good-job ""
     ?f1 <- (goal get-another-point-from-user y)
=>
     (retract ?f1)
     (assert (edit-succeeded user-reports y)))
```

(rules continued)

```
;-------------------------------------------------------
; This section rejects edits. It asks the user if he
;    agrees with the rejection decision.
;-------------------------------------------------------
(defrule reject-edit ""
     ?f1 <- (goal evaluate edit-point)
     ?f2 <- (?expert rejects edit ?reason)
=>
     (printout "Edit rejected by " ?expert " expert
          because " ?reason crlf)
     (retract ?f1)
     (retract ?f2)
     (assert (goal reject-edit)))

(defrule prefer-earlier-and-later ""
     ?f1 <- (goal evaluate edit-point)
     (?expert1 prefers-earlier ?take)
     (?expert2 prefers-later ?take)
=>
     (retract ?f1)
     (assert (goal reject-edit)))

(defrule rejected-edit ""
     ?f1 <- (goal reject-edit)
     ?f2 <- (goal execute-edit)
=>
     (retract ?f1)
     (retract ?f2)
     (printout
          "The editor could not accomplish an
          acceptable edit here." crlf "Were you able to
          complete an acceptable edit(y/n)?")
     (bind ?success (read))
     (assert (edit-failed user-reports ?success)))
```

(rules continued)

```
;-------------------------------------------------------
; This section would trigger learning functions based
;    on the users input. At present, it merely asks the
;    user if he agrees with the output given by the
;    editor and comments on what should be done.
;-------------------------------------------------------
;If we failed, but the user succeeds, we must relax the
;    constraints that caused our failure.
(defrule failed-user-succeeds ""
     ?f1 <- (goal learn-from-edit)
     ?f2 <- (edit-failed user-reports y)
=>
     (retract ?f1)
     (retract ?f2)
     (printout "Either I'm too picky or you're very
          talented." crlf)
     (assert (goal cleanup edit-facts)))

;If we succeeded and the user agrees, pat ourselves on
;    the back. We need to strengthen our current
;    settings (make them harder to change)
(defrule success-user-success ""
     ?f1 <- (goal learn-from-edit)
     ?f2 <- (edit-succeeded user-reports y)
=>
     (retract ?f1)
     (retract ?f2)
     (printout "Am I good or what?" crlf)
     (assert (goal cleanup edit-facts)))

;If we failed and the user did too, console him and
;    strengthen our current settings.
(defrule failed-user-failed ""
     ?f1 <- (goal learn-from-edit)
     ?f2 <- (edit-failed user-reports n)
=>
     (retract ?f1)
     (retract ?f2)
     (printout "Some things just were not meant to be."
          crlf)
     (assert (goal cleanup edit-facts)))
```

(rules continued)

```
;If we succeeded and the user failed, apologize and
;    tighten the constraints.
(defrule success-user-failed ""
     ?f1 <- (goal learn-from-edit)
     ?f2 <- (edit-succeeded user-reports n)
=>
     (retract ?f1)
     (retract ?f2)
     (printout "What was I thinking to propose such a
          bogus edit." crlf " Can you ever forgive
          me?")
     (assert (goal cleanup edit-facts)))

;If we succeeded but the user prefers another edit,
;    examine what is different and alter the
;    constraints.
(defrule success-user-picks-another ""
     ?f1 <- (goal learn-from-edit)
     ?f2 <- (edit-succeeded user-reports another)
=>
     (retract ?f1)
     (retract ?f2)
     (printout "I'm still learning, I'll get it right
          next time." crlf)
     (assert (goal cleanup edit-facts)))

;-------------------------------------------------------
; This section is used to clean up after an edit and
;    prepare to do a new edit.
;-------------------------------------------------------
(defrule remove-takes ""
     (goal cleanup edit-facts)
     ?f1 <- (?take take-is ?name)
=>
     (retract ?f1))

(defrule remove-ranges ""
     (goal cleanup edit-facts)
     ?f1 <- (?take range-is ?start ?stop)
=>
     (retract ?f1))

(defrule remove-edit-planned ""
     (goal cleanup edit-facts)
     ?f1 <- (edit planned)
=>
     (retract ?f1))
```

(rules continued)

```
(defrule remove-sampling-rates ""
     (goal cleanup edit-facts)
     ?f1 <- (?take sampling-rate-is ?value)
=>
     (retract ?f1))

(defrule remove-edit-at ""
     (goal cleanup edit-facts)
     ?f1 <- (?expert ?take ?start ?stop edit-at ?place)

=>
     (retract ?f1))

(defrule remove-low-frequency ""
     (goal cleanup edit-facts)
     ?f1 <- (?take low-frequency-is ?freq)
=>
     (retract ?f1))

(defrule remove-edit-points ""
     (goal cleanup edit-facts)
     ?f1 <- (edit-points-at ?point1 ?point2)
=>
     (retract ?f1))

(defrule remove-edit ""
     (goal cleanup edit-facts)
     ?f1 <- (?take edit-is ?place)
=>
     (retract ?f1))

(defrule remove-envelope-analyzed ""
     (goal cleanup edit-facts)
     ?f1 <- (?expert analyzed ?start ?stop)
=>
     (retract ?f1))

(defrule remove-sampling-rate-match ""
     (goal cleanup edit-facts)
     ?f1 <- (have sampling-rates)
=>
     (retract ?f1))
```

(rules continued)

```
;-------------------------------------------------------
; This section allows the editing process to be
;    terminated or restarted after the completion of an
;    edit.
;-------------------------------------------------------

(defrule ask-about-start-again ""
     (declare (salience -5))
     ?f1 <- (goal cleanup edit-facts)
=>
     (retract ?f1)
     (printout "Do you want to do another edit? (y/n)"
          crlf)
     (bind ?answer (read))
     (assert (user said ?answer)))

(defrule terminate ""
     ?f1 <- (user said n)
=>
     (exit))

(defrule start-again ""
     ?f1 <- (user said y)
=>
     (retract ?f1)
     (assert (goal plan-edit))
     (assert (goal execute-edit))
     (assert (goal learn-from-edit)))
```

APPENDIX B

The Envelope Expert

```
/*******************************************************
*
*                          EXP-ENV.H
*
*******************************************************
*
* These are the defines for the envelope analyzer
*    section of the digital audio editor.
*
*******************************************************/

#define ENVELOPE_NAME          "EXP-env"
#define MAILBOX_PROG           "/editor/bin/mailbox"
#define BLAKBORD_NAME          "blakbord"

#define TRUE                   1
#define FALSE                  0

#define YES                    1
#define NO                     0

#define BACKGROUND             1
#define POLLING_RATE           2

#define UP                     1
#define DOWN                   0

#define SAMPLE_RATE_MSBS       8
#define SAMPLE_RATE_LSBS       2


/*******************************************************
*
*                          GLOBALS.H
*
*******************************************************
*
* These are the global variable declarations for the
*    envelope analyzing expert for the digital audio
*    editor.
*
*******************************************************/

char           incoming_msg_gca[MAX_MESSAGE_LENGTH];
char           directory_name_gca[256];
char           direction_gc = DOWN;
char           take_name_gca[17];
char           error_message_gca[255];
```

(globals.h continued)

```
int                         highest_freq_gi = 20000;
int                         lowest_freq_gi = 100;
int                         crest_maximum_gi[2];
int                         take_number_gi;
int                         terminate_gi = FALSE;
int                         *result_gia;

unsigned                    msg_from_gu;
unsigned                    mailbox_address_gu;
unsigned                    blakbord_gu;
unsigned                    array_length_gu;
unsigned                    array_minimum_gu;
unsigned                    array_min_place_gu;

long                        range_start_gl = 0;
long                        range_end_gl = 0;
long                        *result_place_gla;
long                        sampling_rate_gl = 0;

set_range_mat               *set_range_gmap;
set_take_mat                *set_take_gmap;
set_low_frequency_mat       *set_low_frequency_gmap;
id_only_mat                 id_only_gma;
evaluate_edit_mat           *evaluate_edit_gmap;
assert_mat                  assert_gma;

FILE                        *text_out_gfp;


/*******************************************************
*
*                           EXP-ENV.C
*
/*******************************************************
*
* PURPOSE : This is the main routine. Its job is to
*    parse individual messages to the envelope
*    analyzing expert and call the appropriate
*    functions to perform the task requested in the
*    message.
*
*    PARAMETERS : none
*
*    RETURNS : zero
*
*******************************************************/
#include <string.h>
#include <stdio.h>
#include <process.h>
#include <magic.h>
```

(exp-env.c continued)

```
#include <systids.h>
#include <stdlib.h>
#include "exp-env.h"
#include "/clips/clips/include/messages.h"
#include "globals.h"
#include "protos.h"

main ()

{
     extern    long lowest_freq_gi;

     char      timing_c = 0;
     int       assert_message_i = FALSE;
     int       send_results_i;
     int       message_from_i;
     long      edit_at_l;

/*
** Start text area.
*/
     text_out_gfp = fopen (TEXT_OUTPUT,"w");

/*
** Start a mailbox with my name so that messages and
**   requests can be received.
*/
     mailbox_address_gu = createq
          (BACKGROUND,MAILBOX_PROG,ENVELOPE_NAME,NULL);

     send (mailbox_address_gu,&timing_c,&timing_c,1,1);


/*
** Locate other mailboxes.
*/
     if (! (blakbord_gu = name_locate
          (BLAKBORD_NAME,NAME_SERVER_NODE,
          MAX_MESSAGE_LENGTH)))
     {
          fprintf (text_out_gfp,
               "\nEnvelope expert can't find the
               blackboard.");
          exit (1);
     }
/*
** Tell Decision maker I am ready.
*/
     message_from_i = receive (0,&timing_c,1);
     reply (message_from_i,&timing_c,1);
```

(exp-env.c continued)

```
/*
** Set up request message.
*/
     id_only_gma.struct_id_c = ID_REQUEST_MESSAGE;
/*
** Set up data_locations
*/
     strcpy (directory_name_gca,DATA_DIRECTORY);
/*
** Take messages off of the queue and execute them.
*/
     while (! terminate_gi)
     {
          send (mailbox_address_gu,&id_only_gma,
               incoming_msg_gca, sizeof (id_only_mat),
               sizeof (incoming_msg_gca));

          switch (incoming_msg_gca[0])
          {
               case ID_TERMINATE :
                    terminate_gi = TRUE;
                    break;
               case ID_SET_RANGE :
                    set_range_gmap = (set_range_mat *)
                         incoming_msg_gca;
                    range_start_gl =
                         set_range_gmap ->
                         start_of_range_l;
                    range_end_gl =
                         set_range_gmap ->
                         end_of_range_l;
                    sprintf (assert_gma.assert_mssg_ca,
                         "envelope range-is %ld %ld",
                         range_start_gl,range_end_gl);
                    assert_message_i = TRUE;
                    fprintf (text_out_gfp,
                         "\nEnvelope setting range to
                         %ld %ld.", range_start_gl,
                         range_end_gl);
                    break;
               case ID_GET_SAMPLING_RATE :
                    if (0 < get_sample_rate_lf())
                    {
                         sprintf
                         (assert_gma.assert_mssg_ca,
                              "%s sampling-rate-is
                              %ld", take_name_gca,
                              sampling_rate_gl);
                    } else {
```

(exp-env.c continued)

```
            strcpy
                 (assert_gma.
                 assert_mssg_ca,
                 error_message_gca);
        }
        fprintf (text_out_gfp,
             "\nEnvelope got sampling rate
             of %ld.", sampling_rate_gl);
        assert_message_i = TRUE;
        break;
case ID_SET_TAKE :
        set_take_gmap =
             (set_take_mat *)
             incoming_msg_gca;
        strcpy (take_name_gca,
             set_take_gmap->take_name_ca);
        sprintf (assert_gma.assert_mssg_ca,
             "envelope take-is
             %s",take_name_gca);
        assert_message_i = TRUE;
        fprintf (text_out_gfp,
             "\nEnvelope setting take to
             %s.", take_name_gca);
        break;
case ID_SET_TAKE_NUMBER :
        set_take_gmap =
             (set_take_mat *)
             incoming_msg_gca;
        if (0 == strcmp
             (set_take_gmap->take_name_ca,
             "first"))
        {
             take_number_gi = 0;
        } else {
             take_number_gi = 1;
        }
        break;
case ID_SET_LOW_FREQUENCY :
        set_low_frequency_gmap =
             (set_low_frequency_mat *)
             incoming_msg_gca;
        lowest_freq_gi =
             set_low_frequency_gmap ->
             low_frequency_i;
        fprintf (text_out_gfp,
             "\nEnvelope setting lowest
             frequency to %d.",
             lowest_freq_gi);
```

(exp-env.c continued)

```
        sprintf (assert_gma.assert_mssg_ca,
             "envelope low-frequency-is
             %d", lowest_freq_gi);
        assert_message_i = TRUE;
        break;
   case ID_EVALUATE_EDIT :
        evaluate_edit_gmap =
             (evaluate_edit_mat *)
             incoming_msg_gca;
        strcpy (assert_gma.assert_mssg_ca,
             directory_name_gca);
        strcat (assert_gma.assert_mssg_ca,
             evaluate_edit_gmap ->
             take_name_ca);
        evaluate_edit_point_cf
             (assert_gma.assert_mssg_ca,
             evaluate_edit_gmap ->
             start_of_range_l);
        assert_message_i = TRUE;
        break;
   case ID_EMPTY_REPLY :
        sleep (POLLING_RATE);
        break;
   case ID_ANALYZE :
        if (! analyze_range_if ())
        {
             sprintf
                  (assert_gma.
                  assert_mssg_ca,
                  "envelope analyzed %ld
                  %ld", range_start_gl,
                  range_end_gl);
             assert_message_i = TRUE;
        }
        break;
   case ID_FIND_EDIT :
        if (! (edit_at_l =
             find_edit_lf ()))
        {
             sprintf
                  (assert_gma.
                  assert_mssg_ca,
                  "envelope
                  no-possible-edit-
                  in-range");
        } else {
```

(exp-env.c continued)

```
                    sprintf
                         (assert_gma.
                         assert_mssg_ca,
                         "envelope %s %ld %ld
                         edit-at %ld",
                         take_name_gca,
                         range_start_gl,
                         range_end_gl, edit_at_l);
                    }
                    assert_message_i = TRUE;
                    break;
               default :
                    fprintf (text_out_gfp,
                         "\nERROR -- Envelope analyzer
                         received illegal message.");
                    break;
          }
          if (assert_message_i)
          {
               assert_gma.struct_id_c =
                    ID_ASSERT_MESSAGE;
               send_results_i =
                    send (blakbord_gu, &assert_gma,
                    &assert_gma, sizeof(assert_mat),
                    sizeof(assert_mat));
               assert_message_i = FALSE;
          }
     }
     fclose (text_out_gfp);
     return (0);
}
```

```
/*******************************************************
*
*                          ANALYZER.C
*
*******************************************************
*
* PURPOSE : This routine analyzes the envelope
*    producing is the main routine. Its job is to parse
*    individual messages to the envelope analyzing
*    expert and call the appropriate functions to
*    perform the task requested in the message.
*
*******************************************************/
/*
** WARNING : THIS ROUTINE HAS MULTIPLE RETURNS.
**        This routine returns under error conditions.
*/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "exp-env.h"
#include "protos.

int       analyze_range_if ()

{
     extern    char      directory_name_gca[];
     extern    char      direction_gc;
     extern    char      take_name_gca[17];
     extern    int       highest_freq_gi;
     extern    int       lowest_freq_gi;
     extern    int       take_number_gi;
     extern    int       *result_gia;
     extern    unsigned  array_length_gu;
     extern    long      range_start_gl;
     extern    long      range_end_gl;
     extern    long      *result_place_gla;
     extern    long      sampling_rate_gl;

               char done_c = FALSE;
               char file_name_ca[256];
               char found_new_max_c = FALSE;
               int  second_max_i;
               int  frame_start_i;
               int  window_size_i;
               int  *input_buffer_ia;
               int  last_read_i;
               int  buffer_ptr_i;
     unsigned  int  array_needed_u;
               long read_from_position_l;
               long array_pointer_l;
```

(analyzer.c continued)

```
                long second_max_place_l;
                long window_start_l;
                long backup_one_window_l;
                FILE *source_file_fp;

     if (range_start_gl == 0)
     {
          return (1);
     }

     if (range_end_gl == 0)
     {
          return (2);
     }

     if (sampling_rate_gl == 0)
     {
          return (3);
     }

/*
** Calculate the array which will be generated in the
**   analysis process.
**   NOTE : This system cannot handle note separations
**   of more than 1.4 seconds at 44100 sampling rate.
*/
     array_needed_u =
          ((range_end_gl - range_start_gl) /
           (sampling_rate_gl / highest_freq_gi));
     if (array_needed_u > 64000)
     {
          return (5);
     }

/*
** If an array exists, get rid of it.
*/
     if (result_gia)
     {
          free (result_gia);
          free (result_place_gla);
     }
```

(analyzer.c continued)

```
/*
** Try to acquire an array the appropriate size array
**   for the storage of the results.
*/
     if (! (result_gia =
          (int *) calloc (array_needed_u,
          sizeof (int))))
     {
          return (6);
     }
     if (! (result_place_gla =
          (long *) calloc (array_needed_u,
          sizeof (long))))
     {
          return (7);
     }
     window_size_i = sampling_rate_gl / lowest_freq_gi;

     if (! (input_buffer_ia =
          (int *) calloc (window_size_i,sizeof (int))))

     {
          return (8);
     }

/*
** Set up directory name and structure.
*/
     strcpy (file_name_ca,directory_name_gca);
     strcat (file_name_ca,take_name_gca);
```

(analyzer.c continued)

```
/*
** If the file cannot be opened set an error condition
**   to the decision maker.
*/
     if (! (source_file_fp = fopen (file_name_ca,"r")))

     {
          return (9);
     }

/*
** If the file is open, seek the position for the start
**   of the range.
*/
     fseek (source_file_fp,
          (read_from_position_l =
          range_start_gl * 2L),SEEK_SET);
```

(analyzer.c continued)

```
/*
** Fill array while analyzing read material.
*/
     array_pointer_l = 0;
     frame_start_i = -1;
     result_gia[array_pointer_l] = 0;
     last_read_i = 0;
     second_max_i = 0;
     buffer_ptr_i = window_size_i;

/*
** While we are not done, read a full frame or whatever
**   remains of a frame.
*/
     while (((range_end_gl * 2L) >=
          read_from_position_l)
          && (! done_c))
     {

     /*
     ** Examine the remaining part of the array. If a
     **   new maximum is found, stop and store new max
     **   and set window start to new max. Otherwise,
     **   continue to process until entire window is
     **   processed saving secondary max. Save
     **   secondary max and start new window at
     **   secondary max.
     */
          while ((! found_new_max_c) &&
                  (buffer_ptr_i != frame_start_i) &&
                  (buffer_ptr_i < window_size_i))
          {
               found_new_max_c = look_for_maximums_cf
                    (buffer_ptr_i++,input_buffer_ia,
                    &second_max_i, &last_read_i,
                    &second_max_place_l,
                    array_pointer_l,
                    &read_from_position_l);
          }
```

(analyzer.c continued)

```
/*
** If we found a new max, move the window to the
**   new maximum.
*/
     if (found_new_max_c)
     {
          frame_start_i = buffer_ptr_i - 1;
          second_max_i = 0;
          second_max_place_l = 0;
          found_new_max_c = 0;
          result_gia[array_pointer_l + 1] =
               result_gia[array_pointer_l];
          array_pointer_l++;

/*
** If not a new max, but we finished the buffer,
**   save the secondary.
*/
     } else if (buffer_ptr_i == frame_start_i)
     {
          frame_start_i =
               ((window_size_i + buffer_ptr_i) -
               ((read_from_position_l -
               second_max_place_l) /
               sizeof (int)))
               % window_size_i;
          result_gia[array_pointer_l] =
               second_max_i;
          result_place_gla[array_pointer_l] =
               second_max_place_l;
          result_gia[array_pointer_l + 1] =
               result_gia[array_pointer_l];
          if (frame_start_i > buffer_ptr_i)
          {
               backup_one_window_l = (long)
                    (-2L * (long) window_size_i *
                    (long) sizeof (int));
               fseek (source_file_fp,
                    backup_one_window_l,
                    SEEK_CUR);
               window_start_l =
                    ftell (source_file_fp);
               if (! (window_size_i =
                    fread ((void *)
                    input_buffer_ia, sizeof (int),
                    window_size_i,
                    source_file_fp)))
               {
                    done_c = TRUE;
               }
```

(analyzer.c continued)

```
                }
                read_from_position_l =
                     second_max_place_l + 2;
                buffer_ptr_i = frame_start_i + 1;
                array_pointer_l++;
                second_max_i = 0;
                second_max_place_l = 0;
                direction_gc = DOWN;
           } else if (buffer_ptr_i == window_size_i)
           {
                window_start_l = ftell (source_file_fp);

                if (! (window_size_i =
                     fread ((void *) input_buffer_ia,
                     sizeof (int), window_size_i,
                     source_file_fp)))
                {
                     done_c = TRUE;
                } else {
                     buffer_ptr_i = 0;
                }
           }
      }
      array_length_gu = array_pointer_l - 1;

      return (0);
}
```

```
/*******************************************************
*
*                          FINDEDIT.C
*
*******************************************************
*
* PURPOSE : To analyze the envelope produced by the
*    analyzer to determine an appropriate edit point
*    and extract rate of degradation information.
*
*******************************************************

#include <stdio.h>
#include "exp-env.h"

long find_edit_lf (void)

{
     extern    int       *result_gia;
     extern    int       take_number_gi;
     extern    int       crest_maximum_gi[];
     extern    unsigned  array_length_gu;
     extern    unsigned  array_minimum_gu;
     extern    unsigned  array_min_place_gu;
     extern    long      *result_place_gla;
     extern    FILE      *text_out_gfp;

               int       done_i;
               int       attack_started_i;
               unsigned  count_u;
               long      return_value_l = 0;
               double    average_change_rate_d;
               double    current_rate1_d;
               double    current_rate2_d;

     fprintf (text_out_gfp,"\nEnvelope expert searching
          for edit.");

/*
** Find minimum value in the array.
*/
     array_minimum_gu = (unsigned) 65534;
     for (count_u = 0; count_u <= array_length_gu;
          count_u++)
     {
          if (result_gia[count_u] < array_minimum_gu)
          {
               array_minimum_gu = result_gia[count_u];
               array_min_place_gu = count_u;
          }
     }
```

(findedit.c continued)

```
/*
** Find maximum value for second crest.
*/
     crest_maximum_gi[take_number_gi] = 0;
     for (count_u = array_length_gu / 2;
          count_u <= array_length_gu; count_u++)
     {
          if (result_gia[count_u] >
               crest_maximum_gi[take_number_gi])
          {
               crest_maximum_gi[take_number_gi] =
                    result_gia[count_u];
          }
     }
     fprintf (text_out_gfp,
          "\nEnvelope Expert saving maximum crest value
          of %d for take %d.",
          crest_maximum_gi[take_number_gi],
          take_number_gi);

/*
** Find the place where the slope begins to increase.
**   This is defined as the point where the slope
**   becomes greater than the average rate of increase.
**   This detects sudden changes in amplitude to
**   classify them as note attacks.
*/
     done_i = FALSE;
     count_u = array_length_gu;
     average_change_rate_d =
          (double) (result_gia[array_length_gu] -
          array_minimum_gu) /
          (double) ((result_place_gla[array_length_gu]
-         result_place_gla[0]) / 2);
     attack_started_i = FALSE;
     while ((count_u >= 2) &&
             (! done_i))
     {
          current_rate1_d = (double)
               (result_gia[count_u] -
               result_gia[count_u - 1]) /
               (double) (result_place_gla[count_u] -
               result_place_gla[count_u - 1]);
          current_rate2_d = (double)
               (result_gia[count_u] -
               result_gia[count_u - 2]) /
                (double) (result_place_gla[count_u] -
               result_place_gla[count_u - 2]);
```

(findedit.c continued)

```
            if ((! attack_started_i) &&
                 (average_change_rate_d <
                 current_rate1_d))
            {
                 attack_started_i = TRUE;
            } else if ((attack_started_i) &&
                 (average_change_rate_d >
                 current_rate1_d) &&
                 (average_change_rate_d >
                 current_rate2_d))
            {
                 done_i = TRUE;
                 return_value_l =
                      result_place_gla[count_u - 1] / 2;
            }
            count_u--;
       }
       if ((! count_u) && (! done_i))
       {
            return_value_l = 0;
       }
       return (return_value_l);
}
```

```
/*******************************************************
*
*                          GETSRATE.C
*
*******************************************************
*
* Purpose: to recover the sampling rate information
*    from an Hyperception type file of audio samples.
*
*******************************************************/

#include <stdio.h>
#include <string.h>
#include "/clips/clips/include/messages.h"
#include "getsrate.h"

long get_sample_rate_lf ()

{
     extern    char take_name_gca[];
     extern    char error_message_gca[];
     extern    long sampling_rate_gl;
     extern    FILE *text_out_gfp;

               int  srate_i;
               char take_file_name_ca[256];
               FILE *take_file_fh;

     strcpy (take_file_name_ca,DATA_DIRECTORY);
     strcat (take_file_name_ca,take_name_gca);
     if (! (take_file_fh =
          fopen (take_file_name_ca,"r")))
     {
          strcpy (error_message_gca,"error
               invalid-file-name envelope-expert");
          return (-1L);
     }
     if (fseek (take_file_fh,SAMPLE_RATE_MSBS *
          sizeof (int),SEEK_SET))
     {
          strcpy (error_message_gca,"error
               invalid-file-length envelope-expert");
          return (-2L);
     }
     fread ((void *) &srate_i, sizeof (int), 1,
          take_file_fh);
     sampling_rate_gl = (long) srate_i;
     if (fseek (take_file_fh,
          SAMPLE_RATE_LSBS * sizeof (int), SEEK_SET))
     {
```

(getsrate.c continued)

```
            strcpy (error_message_gca,"error
                 invalid-file-length envelope-expert");
            return (-3L);
        }
        sampling_rate_gl *= 32767;
        fread ((void *) &srate_i, sizeof (int), 1,
            take_file_fh);
        sampling_rate_gl += (long) srate_i;
        fclose (take_file_fh);
        return (sampling_rate_gl);
}
```

```
/*********************************************************
*
*                          EVALUATE.C
*
*********************************************************
*
* This section evaluates edits in terms of amplitudes.
*    Only sufficient methods have been applied for the
*    examples. Methods of improvement include:
*         Checking the value of the change against the
*    value of the next attack which would have occurred
*    had there been no edit.
*         Comparison of envelope immediately preceding
*    and following the edit. If the amplitude drops
*    suddenly because of the edit, the incoming edit
*    can be made later (preferred) or the outgoing edit
*    can be made earlier. If either shift is
*    inappropriate, the next pass will produce a
*    "prefer-earlier" by one expert and a
*    "prefer-later" by another, thus rejecting the
*    edit.
*
*********************************************************/

#include <stdlib.h>
#include <stdio.h>

char *evaluate_edit_point_cf (char * results_cp,
                              long point1_l)

{
     extern    int       crest_maximum_gi[];
     extern    long      sampling_rate_gl;
     extern    FILE      *text_out_gfp;

     int  *buffer_in_ip, *buffer_out_ip;
     int  allowable_decrease_amplitude_i = 5;
     int  allowable_increase_amplitude_i = 30;
     int  allowable_decrease_in_attack_i = 5;
     int  allowable_increase_in_attack_i = 30;
     int  input_max_i = 0, output_max_i = 0;
     int  idx_i;
     long buffer_length_needed_l;
     FILE *output_file_fp;

     fprintf (text_out_gfp,
          "\nEnvelope evaluating edit.");
```

(evaluate.c continued)

```
/*
** All these evaluations should use a case frame
**   parameter. In this case, it is hard coded to be 5%
**   for the purpose of this research.
*/
     if ((crest_maximum_gi[0] > crest_maximum_gi[1]) &&
          ((long) allowable_decrease_in_attack_i <
               ((100L * (long) (crest_maximum_gi[0] -
          crest_maximum_gi[1])) /
          (long) crest_maximum_gi[0])))
     {
          sprintf (results_cp,"envelope rejects edit
               next-attack-amplitude-decrease");
          fprintf (text_out_gfp,"\nEnvelope rejecting
               edit because of attack amplitude
               decrease.");
     }

/*
** Allowable increase in amplitude has been limited to
**   30 percent.
*/
     else if ((crest_maximum_gi[1] >
          crest_maximum_gi[0]) && ((long)
          allowable_increase_in_attack_i <
          ((100L * (long) (crest_maximum_gi[1] -
          crest_maximum_gi[0])) /
          (long) crest_maximum_gi[1])))
     {
          sprintf (results_cp,"envelope rejects edit
               next-attack-amplitude-increase");
          fprintf (text_out_gfp,"\nEnvelope rejecting
               edit because of attack amplitude
               increase.");
     }

/*
** Check amplitudes immediately before and after the
**   edit point.
*/

/*
NOTE : This section is still untested and not used in
the examples.

     else {
          buffer_length_needed_l = (float) (((float)
               sampling_rate_gl) / ((float) 100.0));
```

(evaluate.c continued)

```
buffer_in_ip =
     (int *) calloc
     ((int) buffer_length_needed_l,
     sizeof (int));
buffer_out_ip = (int *) calloc ((int)
     buffer_length_needed_l,
     sizeof (int));
output_file_fp = fopen (results_cp,"r");
fseek (output_file_fp,(point1_l -
     (long) buffer_length_needed_l),
     SEEK_SET);
fread ((void *) buffer_in_ip,
     (int) buffer_length_needed_l,
     sizeof (int), output_file_fp);
fread ((void *) buffer_out_ip,
     (int) buffer_length_needed_l,
     sizeof (int), output_file_fp);
fclose (output_file_fp);
for (idx_i = 0; idx_i <
     buffer_length_needed_l; idx_i++)
{
     if (input_max_i < buffer_in_ip[idx_i])
     {
          input_max_i = buffer_in_ip[idx_i];
     }
     if (output_max_i < buffer_out_ip[idx_i])
     {
          output_max_i =
               buffer_out_ip[idx_i];
     }
}
if (output_max_i < (input_max_i *
     allowable_decrease_amplitude_i))
{
     sprintf (results_cp,"envelope
          prefers-later second");
}
else if (output_max_i > (input_max_i *
     allowable_increase_amplitude_i))
{
     sprintf (results_cp,"envelope
          prefers-later first");
}
*/
```

(evaluate.c continued)

```
/*
** If there is no known reason to reject the edit,
**   accept it.
*/
     else {
          fprintf (text_out_gfp,"\nEnvelope accepting
               edit.");
          sprintf (results_cp,"envelope accepts edit");
     }
     return (results_cp);
}
```

APPENDIX C

The Spectrum Expert

```
/*********************************************************
*
*                          EXP-FFT.H
*
*********************************************************
*
* These are the defines for the envelope analyzer
*    section of the digital audio editor.
*
*********************************************************/

#define SPECTRUM_NAME     "EXP-fft"
#define MAILBOX_PROG      "/editor/bin/mailbox"
#define BLAKBORD_NAME     "blakbord"

#define TRUE              1
#define FALSE             0

#define YES               1
#define NO                0

#define BACKGROUND        1
#define POLLING_RATE      2

#define WINDOW_SIZE       2048
#define DELTA_SIZE        50
#define NUMBER_MAXES      10
```

```
/*******************************************************
*
*                          GLOBALS.H
*
*******************************************************
*
* These are the global variable declarations for the
*    envelope analyzing expert for the digital audio
*    editor.
*
*******************************************************/

char            incoming_msg_gca[MAX_MESSAGE_LENGTH];
char            directory_name_gca[256];
char            take_name_gca[17];
char            error_message_gca[255];

int             highest_frequency_gi = 10000;
int             lowest_frequency_gi = 100;
int             terminate_gi = FALSE;
int             *result_gia;

unsigned        msg_from_gu;
unsigned        mailbox_address_gu;
unsigned        blakbord_gu;

long            range_start_gl = 0;
long            range_end_gl = 0;
long            sampling_rate_gl = 0;
long            window_size_gl = 100;

float           harmonic_separation_gf;

sample_rate_mat     *sample_rate_gmap;
set_range_mat       *set_range_gmap;
set_take_mat        *set_take_gmap;
id_only_mat         id_only_gma;
assert_mat          assert_gma;

FILE                *text_out_gfp;
```

```
/*******************************************************
*
*                          PROTOS.H
*
*******************************************************
*
* These are the prototypes for functions used by the
*    spectrum analyzer
*
*******************************************************/

int       compare_maxes_if (float old_maxes_fa[][],
                            float new_maxes_fa[][]);

long find_edit_lf (void);

void find_maxes_vf (float * data, float maxes[][]);

void FFTCalc (realtype *xreal, realtype *yimag,
              int numdat);

long get_sample_rate_lf (void);
```

```
/*******************************************************
*
*                          EXP-FFT.C
*
*******************************************************
*
* PURPOSE : This is the main routine. Its job is to
*    parse individual messages to the envelope
*    analyzing expert and call the appropriate
*    functions to perform the task requested in the
*    message.
*
*    PARAMETERS : none
*
*    RETURNS : zero
*
*******************************************************/
#include <string.h>
#include <stdio.h>
#include <process.h>
#include <magic.h>
#include <systids.h>
#include <stdlib.h>
#include "exp-fft.h"
#include "/clips/clips/include/messages.h"
#include "globals.h"
#include "realtype.h"
#include "protos.h"

main ()

{
     char timing_c = 0;
     int       assert_message_i = FALSE;
     int       send_results_i;
     int       message_from_i;
     long edit_l;

/*
** Start an output for text.
*/
     text_out_gfp = fopen (TEXT_OUTPUT,"w");

/*
** Start a mailbox with my name so that messages and
**   requests can be received.
*/
     mailbox_address_gu =
          createq (BACKGROUND, MAILBOX_PROG,
          SPECTRUM_NAME, NULL);
     send (mailbox_address_gu,&timing_c,&timing_c,1,1);
```

(exp-fft.c continued)

```
/*
** Locate other mailboxes.
*/
     if (! (blakbord_gu =
          name_locate (BLAKBORD_NAME,NAME_SERVER_NODE,
          MAX_MESSAGE_LENGTH)))
     {
          fprintf (text_out_gfp,"\nSpectrum expert
               can't find the blackboard.");
          exit (1);
     }

/*
** Tell decision maker that I am ready.
*/
     message_from_i = receive (0,&timing_c,1);
     reply (message_from_i,&timing_c,1);

/*
** Set up request message.
*/
     id_only_gma.struct_id_c = ID_REQUEST_MESSAGE;

/*
** Set up data_locations
*/
     strcpy (directory_name_gca,DATA_DIRECTORY);

/*
** Take messages off of the queue and execute them.
*/
     while (! terminate_gi)
     {
          send (mailbox_address_gu, &id_only_gma,
               incoming_msg_gca, sizeof (id_only_mat),
               sizeof (incoming_msg_gca));
          switch (incoming_msg_gca[0])
          {
               case ID_TERMINATE :
                    terminate_gi = TRUE;
                    break;
               case ID_SET_RANGE :
                    fprintf (text_out_gfp,"\nSpectrum
                         setting range.");
                    set_range_gmap = (set_range_mat *)
                         incoming_msg_gca;
                    range_start_gl = set_range_gmap ->
                         start_of_range_l;
                    range_end_gl = set_range_gmap ->
                         end_of_range_l;
```

(exp-fft.c continued)

```
        sprintf (assert_gma.assert_mssg_ca,
             "spectrum range-is %ld %ld",
             range_start_gl,range_end_gl);
        assert_message_i = TRUE;
        break;
   case ID_SET_SAMPLING_RATE :
        fprintf (text_out_gfp,
             "\nSpectrum getting sampling
             rate.");
        sample_rate_gmap =
             (sample_rate_mat *)
             incoming_msg_gca;
        sampling_rate_gl =
             sample_rate_gmap -> rate_l;
        sprintf (assert_gma.assert_mssg_ca,
             "spectrum sampling-rate-is
             %ld", sampling_rate_gl);
             harmonic_separation_gf =
             (float) ((float) WINDOW_SIZE *
             (float) (1.0 / (float)
             sampling_rate_gl));
        assert_message_i = TRUE;
        break;
   case ID_SET_TAKE :
        fprintf (text_out_gfp,
             "\nSpectrum setting take.");
        set_take_gmap = (set_take_mat *)
             incoming_msg_gca;
        strcpy (take_name_gca,
             set_take_gmap->take_name_ca);
        sprintf (assert_gma.assert_mssg_ca,
             "spectrum take-is %s",
             take_name_gca);
        assert_message_i = TRUE;
        break;
   case ID_FIND_EDIT :
        fprintf (text_out_gfp,
             "\nSpectrum searching for an
             edit.");
        if (edit_l = find_edit_lf ())
        {
             sprintf
                  (assert_gma.
                  assert_mssg_ca,
                  "spectrum %s %ld %ld
                  edit-at %ld",
                  take_name_gca,
                  range_start_gl,
                  range_end_gl, edit_l);
        } else {
```

(exp-fft.c continued)

```
                    sprintf
                         (assert_gma.
                         assert_mssg_ca,
                         "spectrum
                         no-possible-
                         edit-in-range");
               }
               assert_message_i = TRUE;
               break;
          case ID_GET_LOW_FREQUENCY :
               fprintf (text_out_gfp,
                    "\nSpectrum posting lowest
                    frequency.");
               sprintf (assert_gma.assert_mssg_ca,
                    "spectrum low-frequency-is
                    %d", lowest_frequency_gi);
               assert_message_i = TRUE;
               break;
          case ID_EVALUATE_EDIT :
/*
** This section is not working properly. Edits are
**   automatically accepted by the spectrum analysis at
**   this time.
*/
               fprintf (text_out_gfp,
                    "\nSpectrum evaluating edit
                    point");
               fprintf (text_out_gfp,
                    "\nSpectrum accepting edit.");
               sprintf (assert_gma.assert_mssg_ca,
                    "spectrum accepts edit");
               assert_message_i = TRUE;
               break;
          case ID_EMPTY_REPLY :
               sleep (POLLING_RATE);
               break;
          default :
               fprintf (text_out_gfp,
                    "\nERROR -- Spectrum analyzer
                    received illegal message.");
               break;
     }
     if (assert_message_i)
     {
          assert_gma.struct_id_c =
               ID_ASSERT_MESSAGE;
          send_results_i = send (blakbord_gu,
               &assert_gma, &assert_gma,
               sizeof(assert_mat),
               sizeof(assert_mat));
```

(exp-fft.c continued)

```
                assert_message_i = FALSE;
            }
        }
        return (0);
}
```

```
/*******************************************************
*
*                          COMPMAXS.C
*
*******************************************************
*
* Purpose: This routine compares the maximum values
*    placed in the arrays to see how many of the values
*    are common to both arrays, the values do not have
*    to be in any specific order for this comparison.
*    As long as the value exists in both arrays, it is
*    considered to be a common value.
*
*******************************************************/

#include <stdio.h>
#include "exp-fft.h"

int  compare_maxes_if (
          float old_maxes_fa[NUMBER_MAXES][2],
          float new_maxes_fa[NUMBER_MAXES][2])

{
     extern    FILE *text_out_gfp;

     int       return_value_i = 0;
     int       old_i, new_i;
     int       max_not_found_i;

     fprintf (text_out_gfp,
          "\nSpectrum comparing maximums.");
     for (old_i = 0; old_i < NUMBER_MAXES; old_i++)
     {
          new_i = 0;
          max_not_found_i = 1;
          while ((new_i < NUMBER_MAXES) &&
               (max_not_found_i))
          {
               if ((old_maxes_fa[old_i][1] >
                    new_maxes_fa[new_i][1] - 2) &&
                    (old_maxes_fa[old_i][1] <
                    new_maxes_fa[new_i][1] + 2))
               {
                    max_not_found_i = 0;
               }
               new_i++;
          }
          if (max_not_found_i)
          {
               return_value_i++;
          }
     }
```

(compmaxs.c continued)

```
    return (return_value_i);
}
```

```
/*******************************************************
*
*                          FINDMAXS.C
*
*******************************************************
*
* Purpose: This routine takes an FFT and produces an
*    array of the top NUMBER_MAXES maximum values and
*    their frequencies to be used for comparison and
*    low frequency information.
*
*******************************************************/
#include <math.h>
#include <stdio.h>
#include "exp-fft.h"

void find_maxes_vf (float * data_fp,
                    float maxes_fp[NUMBER_MAXES][2])

{
     extern    int       lowest_frequency_gi;
     extern    long      sampling_rate_gl;
     extern    float     harmonic_separation_gf;
     extern    FILE      *text_out_gfp;

               int       count_i;
               int       index_i;
               int       revers_i;
               int       not_max_i;
               int       new_frequency_i;

     fprintf (text_out_gfp,
          "\nSpectrum analyzing FFT looking for
          maximums.");
     for (count_i = 0; count_i < NUMBER_MAXES;
          count_i++)
     {
          maxes_fp [count_i][0] = 0.0;
     }
     for (count_i = 0; count_i < (WINDOW_SIZE / 2);
          count_i++)
     {
          index_i = 0;
          not_max_i = 1;
          while ((index_i < NUMBER_MAXES) &&
               (not_max_i))
          {
               if (fabs(data_fp[count_i]) >
                    maxes_fp[index_i][0])
               {
                    not_max_i = 0;
                    revers_i = NUMBER_MAXES - 1;
```

(findmaxs.c continued)

```
                while (revers_i > index_i)
                {
                     maxes_fp[revers_i][0] =
                          maxes_fp[revers_i-1][0];
                     maxes_fp[revers_i][1] =
                          maxes_fp[revers_i-1][1];
                     revers_i--;
                }
                maxes_fp[index_i][0] =
                     fabs(data_fp[count_i]);
                maxes_fp[index_i][1] = count_i;
           }
           index_i++;
      }
 }
 fprintf (text_out_gfp,
      "\nSpectrum looking for lowest frequency.");
 new_frequency_i = 22000;
 for (count_i = 0; count_i < NUMBER_MAXES;
      count_i++)
 {
      if (new_frequency_i >
           maxes_fp[count_i][1] /
           harmonic_separation_gf)
      {
           new_frequency_i =
                maxes_fp[count_i][1] /
                harmonic_separation_gf;
      }
 }
 if (lowest_frequency_gi < new_frequency_i)
 {
      fprintf (text_out_gfp,
           "\nSpectrum found higher lowest
           frequency = %d.", new_frequency_i);
      lowest_frequency_gi = new_frequency_i;
 }
 return;
}
```

```
/*******************************************************
*
*                          FINDEDIT.C
*
*******************************************************
*
* The function "FFTCalc" and the include file
*    "realtype.h" are taken from "Software for Science,
*    Engineering and Industry" rev7 Quinn-Curtis
*    21 Highland Circle
*    Needham, MA 02194 USA
*******************************************************
*
* Purpose: This routine locates the point in time where
*    there have been as many high points in the
*    spectrum change as there are high points recorded.
*    This represents the place where a completely new
*    set of spectral components exists and hence is the
*    beginning of a new note.
*
*******************************************************/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "exp-fft.h"
#include "realtype.h"
#include "protos.h"

long find_edit_lf ()

{
     extern    char take_name_gca[];
     extern    char directory_name_gca[];
     extern    long range_start_gl;
     extern    long range_end_gl;
     extern    FILE *text_out_gfp;

     char      file_name_ca[256];
     int       *data_ip;
     int       i;
     int       changed_maxes_i;
     long      current_location_l;
     long      return_value_l;
     float     *real_fp,*imag_fp;
     float     top_maxes_fa[NUMBER_MAXES][2];
     float     last_maxes_fa[NUMBER_MAXES][2];
     FILE      *data_fhp;

     strcpy (file_name_ca,directory_name_gca);
     strcat (file_name_ca,take_name_gca);
```

(findedit.c continued)

```
data_fhp = fopen (file_name_ca,"r");
current_location_l =
     range_end_gl - (WINDOW_SIZE / 2);
changed_maxes_i = 0;
data_ip = (int *) calloc (WINDOW_SIZE,
     sizeof (int));
real_fp = (float *) calloc
     (WINDOW_SIZE,sizeof (float));
imag_fp = (float *) calloc
     (WINDOW_SIZE,sizeof (float));
fseek (data_fhp,current_location_l,SEEK_SET);
fread ((void *) data_ip, WINDOW_SIZE,
     sizeof (int), data_fhp);
for (i = 0; i < WINDOW_SIZE; i++)
{
     real_fp[i] =
          (float) ((float) data_ip[i])/32767.0;
     imag_fp[i] = 0.0;
}
fprintf (text_out_gfp,
     "\nSpectrum doing FFT analysis.");
FFTCalc (real_fp, imag_fp, WINDOW_SIZE);
find_maxes_vf (real_fp,last_maxes_fa);
while ((changed_maxes_i < NUMBER_MAXES) &&
     (current_location_l >= range_start_gl))
{
     current_location_l -= DELTA_SIZE;
     fseek (data_fhp,current_location_l,SEEK_SET);
     fread ((void *) data_ip, WINDOW_SIZE,
          sizeof (int), data_fhp);
     for (i = 0; i < WINDOW_SIZE; i++)
     {
          real_fp[i] = (float)
               ((float) data_ip[i])/32767.0;
          imag_fp[i] = 0.0;
     }
     fprintf (text_out_gfp,
          "\nSpectrum doing FFT analysis.");
     FFTCalc (real_fp, imag_fp, WINDOW_SIZE);
     find_maxes_vf (real_fp,top_maxes_fa);
     changed_maxes_i += compare_maxes_if
          (last_maxes_fa,top_maxes_fa);
     memcpy (last_maxes_fa, top_maxes_fa,
          sizeof (top_maxes_fa));
}
if (current_location_l < range_start_gl)
{
     return_value_l = 0;
} else {
```

(findedit.c continued)

```
            return_value_l =
                 current_location_l + (WINDOW_SIZE / 2);
        }
        free (data_ip);
        free (real_fp);
        free (imag_fp);
        fclose (data_fhp);
        return (return_value_l);
}
```

APPENDIX D

The Rhythm Expert

```
/*********************************************************
*
*                          EXP-RHY.H
*
*********************************************************
*
* These are the defines for the rhythm analyzer section
*    of the digital audio editor.
*
*********************************************************/


#define RHYTHM_NAME             "EXP-rhy"
#define MAILBOX_PROG            "/editor/bin/mailbox"
#define BLAKBORD_NAME           "blakbord"

#define TRUE                    1
#define FALSE                   0

#define YES                     1
#define NO                      0

#define BACKGROUND              1
#define POLLING_RATE            2

#define FIRST                   "first"
#define SECOND                  "second"
```

```
/*******************************************************
*
*                          GLOBALS.H
*
*******************************************************
*
* These are the global variable declarations for the
*    rhythm analyzing expert for the digital audio
*    editor.
*
*******************************************************/

/*
** Space to hold a generic incoming message.
*/
char      incoming_msg_gca[MAX_MESSAGE_LENGTH];

/*
** The name of the current take that is being operated
**   on. All functions in which a take name is assumed
**   are applied to this name.
*/
char      take_name_gca[17];

/*
** Flag to stop message parsing loop.
*/
int       terminate_gi = FALSE;

/*
** Addresses to other tasks.
*/
unsigned  msg_from_gu;
unsigned  mailbox_address_gu;
unsigned  blakbord_gu;

/*
** The range array for each of the two source edit
**   points.
*/
long      range_start_gl[2];
long      range_end_gl[2];

/*
** An array for the two edit points.
*/
long      edit_point_gl[2];
```

(globals.h continued)

```
/*
** Message structures to use as templates for incoming
**   messages and hold data for outgoing messages.
*/

set_range_mat       *set_range_gmap;
set_take_mat        *set_take_gmap;
id_only_mat         id_only_gma;
assert_mat          assert_gma;
set_edit_to_mat     *set_edit_to_gmap;
evaluate_edit_mat   *evaluate_edit_gmap;

/*
** All non-user text messages, trace messages, timing
**   messages, debug messages are sent to this file.
*/

FILE                *text_out_gfp;
```

```
/*******************************************************
*
*                          PROTOS.H
*
*******************************************************
*
* These are the function prototypes for the rhythm
*    analyzing expert.
*
*******************************************************/

char      *evaluate_edit_point_cf (char * results_cp,
                                   long edit_point1_l,
                                   long edit_point2_l);

long      find_edit_lf (void);
```

```
/*******************************************************
*
*                          EXP-RHY.C
*
*******************************************************
*
* PURPOSE : This is the main routine. Its job is to
*    parse individual messages to the rhythm analyzing
*    expert and call the appropriate functions to
*    perform the task requested in the message.
*
*    PARAMETERS : none
*
*    RETURNS : zero
*
*******************************************************/
#include <string.h>
#include <stdio.h>
#include <process.h>
#include <magic.h>
#include <systids.h>
#include <stdlib.h>

#include "exp-rhy.h"
#include "/clips/clips/include/messages.h"
#include "globals.h"
#include "protos.h"

main ()

{
     char timing_c = 0;
     int       assert_message_i = FALSE;
     int       send_results_i;
     int       message_from_i;
     int       take_number_i;

/*
** Start an output for text.
*/
     text_out_gfp = fopen (TEXT_OUTPUT,"w");

/*
** Start a mailbox with my name so that messages and
**   requests can be received. The send is used for
**   timing to insure that the mailbox is running and
**   has registered its name before this program will
**   unblock
*/
     mailbox_address_gu =
          createq (BACKGROUND, MAILBOX_PROG,
               RHYTHM_NAME, NULL);
```

(exp-rhy.c continued)

```
     send (mailbox_address_gu,&timing_c,&timing_c,1,1);

/*
** Locate other mailboxes.
*/
     if (! (blakbord_gu = name_locate (BLAKBORD_NAME,
          NAME_SERVER_NODE, MAX_MESSAGE_LENGTH)))
     {
          fprintf (text_out_gfp,
               "\nSpectrum expert can't find the
               blackboard.");
          exit (1);
     }

/*
** Tell decision maker that I am ready.
*/
     message_from_i = receive (0,&timing_c,1);
     reply (message_from_i,&timing_c,1);

/*
** Set up request message.
*/
     id_only_gma.struct_id_c = ID_REQUEST_MESSAGE;

/*
** Take messages off of the queue and execute them.
*/
     while (! terminate_gi)
     {
          send (mailbox_address_gu, &id_only_gma,
               incoming_msg_gca, sizeof (id_only_mat),
               sizeof (incoming_msg_gca));
          switch (incoming_msg_gca[0])
          {
               case ID_TERMINATE :
                    terminate_gi = TRUE;
                    break;
               case ID_SET_RANGE :
                    set_range_gmap = (set_range_mat *)
                         incoming_msg_gca;
                    range_start_gl[take_number_i] =
                         set_range_gmap ->
                         start_of_range_l;
                    range_end_gl[take_number_i] =
                         set_range_gmap ->
                         end_of_range_l;
```

(exp-rhy.c continued)

```
        sprintf (assert_gma.assert_mssg_ca,
             "rhythm range-is %ld %ld",
             range_start_gl[take_number_i],
             range_end_gl[take_number_i]);
        assert_message_i = TRUE;
        fprintf (text_out_gfp,
             "\nRhythm setting range to %ld
             %ld.",
             range_start_gl[take_number_i],
             range_end_gl[take_number_i]);
        break;
    case ID_SET_TAKE_NUMBER :
        set_take_gmap = (set_take_mat *)
             incoming_msg_gca;
        strcpy (take_name_gca,
             set_take_gmap->take_name_ca);
        if (0 == strcmp (take_name_gca,
             FIRST))
        {
             take_number_i = 0;
        } else if (0 == strcmp
             (take_name_gca, SECOND))
        {
             take_number_i = 1;
        } else {
             sprintf
                  (assert_gma.
                  assert_mssg_ca,
                  "error
                  invalid-take-number
                  rhythm-expert");
        }
        assert_message_i = TRUE;
        fprintf (text_out_gfp,
             "\nRhythm setting take to %s
             take.", take_name_gca);
        break;
    case ID_SET_EDIT_TO :
        set_edit_to_gmap =
             (set_edit_to_mat *)
             incoming_msg_gca;
        edit_point_gl[0] =
             set_edit_to_gmap -> point_1;
        sprintf (assert_gma.assert_mssg_ca,
             "rhythm %s %ld %ld edit-at
             %ld", take_name_gca,
             range_start_gl[0],
             range_end_gl[0],
             edit_point_gl[0]);
        assert_message_i = TRUE;
```

(exp-rhy.c continued)

```
            fprintf (text_out_gfp,
                 "\nRhythm setting edit from
                 source 1 to %ld",
                 edit_point_gl[0]);
            break;
       case ID_FIND_EDIT :
            if (find_edit_lf ())
            {
                 sprintf
                      (assert_gma.
                      assert_mssg_ca,
                      "rhythm %s %ld %ld
                      edit-at %ld",
                      take_name_gca,
                      range_start_gl[1],
                      range_end_gl[1],
                      edit_point_gl[1]);
            } else {
                 sprintf
                      (assert_gma.
                      assert_mssg_ca,
                      "rhythm
                      no-possible-edit-
                      in-range");
            }
            assert_message_i = TRUE;
            break;
       case ID_EVALUATE_EDIT :
            evaluate_edit_gmap =
                 (evaluate_edit_mat *)
                 incoming_msg_gca;
            evaluate_edit_point_cf
                 (assert_gma.assert_mssg_ca,
                 evaluate_edit_gmap ->
                 start_of_range_1,
                 evaluate_edit_gmap ->
                 end_of_range_1);
            assert_message_i = TRUE;
            break;
       case ID_EMPTY_REPLY :
            sleep (POLLING_RATE);
            break;
       default :
            fprintf (text_out_gfp,
                 "\nERROR -- Rhythm analyzer
                 received illegal message.");
            break;
}
```

(exp-rhy.c continued)

```
        if (assert_message_i)
        {
             assert_gma.struct_id_c =
                  ID_ASSERT_MESSAGE;
             send_results_i =
                  send (blakbord_gu, &assert_gma,
                  &assert_gma, sizeof(assert_mat),
                  sizeof(assert_mat));
             assert_message_i = FALSE;
        }
   }
   return (0);
}
```

```
/*******************************************************
*
*                          EVALUATE.C
*
*******************************************************
*
* Purpose: This routine evaluates a completed edit to
*    make sure that the new second crest is within the
*    crossfade time of the place the old second crest
*    occupied.
*
*******************************************************/

#include <stdlib.h>
#include <stdio.h>

char *evaluate_edit_point_cf (char * results_cp,
                              long point1_l,
                              long point2_l)

{
     extern    char take_name_gca[];
     extern    long range_start_gl[];
     extern    long range_end_gl[];
     extern    long edit_point_gl[];
     extern    FILE *text_out_gfp;

               long total_range_length_l;
               long distance_to_next_beat1_l;
               long distance_to_next_beat2_l;
               long percentage_shift_l;

     fprintf (text_out_gfp,
          "\nRhythm evaluating edit.");
     total_range_length_l =
          range_end_gl[0] - range_start_gl[0];
     distance_to_next_beat1_l =
          range_end_gl[0] - point1_l;
     distance_to_next_beat2_l =
          range_end_gl[1] - point2_l;
     percentage_shift_l =
          (100 * labs
          (distance_to_next_beat1_l -
          distance_to_next_beat2_l)) /
          total_range_length_l;
```

(evaluate.c continued)

```
/*
** The percentage of shift should be a variable
**   parameter based on the case frame. It is hard
**   coded here to 5%.
*/
     if (percentage_shift_1 < 5)
     {
          fprintf (text_out_gfp,
               "\nRhythm accepting edit.");
          sprintf (results_cp,"rhythm accepts edit");
     } else {
          if (distance_to_next_beat1_1 >
               distance_to_next_beat2_1)
          {
               fprintf (text_out_gfp,
                    "\nRhythm requesting first source
                    edit be later.");
               sprintf (results_cp,
                    "rhythm prefers-later first");
          } else {
               fprintf (text_out_gfp,
                    "\nRhythm requesting first source
                    edit be earlier.");
               sprintf (results_cp,
                    "rhythm prefers-earlier first");
          }
     }
     return (results_cp);
}
```

```
/*********************************************************
*
*                          FINDEDIT.C
*
*********************************************************
*
* This routine predicts the edit point for the second
*    source based on the edit point for the first
*    source. This assures that there is another attack
*    at the same point after the edit as the next
*    attack was before the edit was performed.
*
*********************************************************/

long      find_edit_1f ()

{
     extern    long range_start_gl[];
     extern    long range_end_gl[];
     extern    long edit_point_gl[];

     long return_value_l = -1L;

     return_value_l = range_end_gl[1] -
          (range_end_gl[0] - edit_point_gl[0]);
     edit_point_gl[1] = return_value_l;
     return (return_value_l);
}
```

APPENDIX E

The Blackboard

```
/*********************************************************
*
*                          BLAKBORD.H
*
*********************************************************/

#define BLACKBOARD_NAME        "blakbord"

struct linked_list_at
{
     char                              struct_id_c;
     char                              name_ca[255];
     struct    linked_list_at          *next_ap;
};

typedef        struct    linked_list_at an_element_at;
typedef        an_element_at            *link_ap;


/*********************************************************
*
*                          GLOBALS.H
*
*********************************************************/

unsigned                  sender_gu;

link_ap                   list_top_ap = NULL;
link_ap                   list_bottom_ap = NULL;

id_only_mat               empty_reply_gma;

FILE                      *text_out_gfp;

/*********************************************************
*
*                          PROTOS.H
*
*********************************************************/


void add_message_to_queue_vf (link_ap new_message_ap);

void post_a_message_vf (void);
```

```
/*******************************************************
*
*                          BLAKBORD.C
*
*******************************************************
*
* Purpose: This program acts as a generic message queue
*    holding messages from the various attached experts
*    to the decision making program. Any message not a
*    request for information is considered new
*    information and is therefore placed on the queue.
*
*******************************************************/

#include <systids.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include "blakbord.h"
#include "/clips/clips/include/messages.h"
#include "globals.h"
#include "protos.h"

main ()

{
     extern          FILE      *text_out_gfp;

     link_ap         new_message_ap;

/*
** Start text area.
*/
     text_out_gfp = fopen (TEXT_OUTPUT,"w");

/*
** Attach name so that other experts can send to the
**   blackboard.
*/
     if (! name_attach (BLACKBOARD_NAME,
          NAME_SERVER_NODE))
     {
          fprintf (text_out_gfp,
               "\nBLACKBOARD -- I can't attach my
               name");
          exit (1);
     }
```

(blakbord.c continued)

```
/*
** Main loop -- receive messages and reply to them. If
**   it is a new assert, add it to the queue. If it is
**   a request for messages from the decision maker,
**   reply with the next message.
*/
     while (1)
     {

     /*
     ** Make a place to hold the incoming message.
     */
          if (! (new_message_ap =
               (link_ap) calloc
               (1,sizeof(an_element_at))))
          {
               fprintf (text_out_gfp,
                    "\nBLACKBOARD ERROR -- ");
               fprintf (text_out_gfp,
                    "\n I can't get memory to
                    accept new message.");
               exit (1);
          }

     /*
     ** Block until a new message is available.
     */
          sender_gu = receive (0,
               (void *) new_message_ap,
               MAX_MESSAGE_LENGTH);

     /*
     ** Check message ID to see if the message is a
     **   request from the decision maker for any ready
     **   asserts.
     */
          if (new_message_ap->struct_id_c ==
               ID_REQUEST_MESSAGE)
          {
               post_a_message_vf ();

     /*
     ** If it is not a request for a message, it must
     **   be a message. Add it to the queue.
     */
          } else {
               add_message_to_queue_vf
                    (new_message_ap);
          }
     }
```

(blakbord.c continued)

```
    fclose (text_out_gfp);

    return (0);
}
```

```
/*********************************************************
*
*                          ADDMESSG.C
*
*********************************************************/

#include <stddef.h>
#include <systids.h>
#include "blakbord.h"
#include "/clips/clips/include/messages.h"

void add_message_to_queue_vf (link_ap new_message_ap)

{
     extern    unsigned            sender_gu;
     extern    link_ap             list_top_ap;
     extern    link_ap             list_bottom_ap;
     extern    id_only_mat         empty_reply_gma;

     new_message_ap -> next_ap = NULL;
     if (list_top_ap)
     {
          list_bottom_ap->next_ap = new_message_ap;
     } else {
          list_top_ap = new_message_ap;
     }
     list_bottom_ap = new_message_ap;
     reply (sender_gu,&empty_reply_gma,
          sizeof (id_only_mat));
     return;
}
```

```
/*********************************************************
*
*                          SENDMSSG.C
*
*********************************************************/

#include <stdlib.h>
#include <systids.h>
#include "blakbord.h"
#include "/clips/clips/include/messages.h"

post_a_message_vf ()

{
     extern    unsigned        sender_gu;
     extern    link_ap         list_top_ap;
     extern    link_ap         list_bottom_ap;
     extern    id_only_mat     empty_reply_gma;

               link_ap         released_mssg_ap;

     if (list_top_ap)
     {
          reply (sender_gu,
               (void *) list_top_ap,
               MAX_MESSAGE_LENGTH);
          released_mssg_ap = list_top_ap;
          list_top_ap = list_top_ap -> next_ap;
          if (! list_top_ap)
          {
               list_bottom_ap = NULL;
          }
          free (released_mssg_ap);
     } else {
          empty_reply_gma.struct_id_c = ID_EMPTY_QUEUE;
          reply (sender_gu,&empty_reply_gma,
               sizeof (id_only_mat));
     }

     return;
}
```

APPENDIX F

The Mailboxes

```
/*********************************************************
*
*                          MAILBOX.H
*
*********************************************************/

#define BLACKBOARD_NAME        "blakbord"

struct linked_list_at
{
     char                              struct_id_c;
     char                              name_ca[255];
     struct    linked_list_at          *next_ap;
};

typedef   struct    linked_list_at an_element_at;
typedef   an_element_at  *link_ap;

/*********************************************************
*
*                          GLOBALS.H
*
*********************************************************/

unsigned             sender_gu;

link_ap              list_top_ap = NULL;
link_ap              list_bottom_ap = NULL;

id_only_mat          empty_reply_gma;

FILE                 *text_out_gfp;

/*********************************************************
*
*                          PROTOS.H
*
*********************************************************/

void add_message_to_queue_vf (link_ap new_message_ap);

void post_a_message_vf (void);
```

```
/*******************************************************
*
*                          MAILBOX.C
*
*******************************************************
*
* Purpose: This program serves as a general message
*    queue for the expert that spawned it. Its first
*    argument contains the name of the expert for which
*    it should receive messages. The messages can be
*    from anyone, though the current implementation
*    should only be sending messages to the experts
*    from the decision maker.
*
*******************************************************/

#include <systids.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include "mailbox.h"
#include "/clips/clips/include/messages.h"
#include "globals.h"
#include "protos.h"

main (int argc, char **argv)

{
     extern          FILE      *text_out_gfp;

     char            timing_c = 0;
     int             message_from_i;
     link_ap         new_message_ap;

/*
** Start text area.
*/
     text_out_gfp = fopen (TEXT_OUTPUT,"w");

/*
** Check parameters.
*/
     if (argc != 2)
     {
          fprintf (text_out_gfp,
               "\nMailbox error. Wrong parameter
               count.");
          exit (1);
     }
```

(mailbox.c continued)

```
/*
** Attach name so that other tasks can send to the
**   mailbox.
*/
     if (! name_attach (argv[1],NAME_SERVER_NODE))
     {
          fprintf (text_out_gfp,
               "\nMAILBOX -- I can't attach the name
               %s.", argv[1]);
          exit (1);
     }

/*
** Tell generating task that I am ready.
*/
     message_from_i = receive (0,
          (void *) &timing_c,sizeof (char));
     reply (message_from_i,&timing_c,1);

/*
** Main loop -- receive messages and reply to them. If
**   it is a new assert add it to the queue. If it is a
**   request for messages from the decision maker,
**   reply with the next message.
*/
     while (1)
     {

     /*
     ** Make a place to hold the incoming message.
     */
          if (! (new_message_ap = (link_ap) calloc
               (1,sizeof(an_element_at))))
          {
               fprintf (text_out_gfp,
                    "\nBLACKBOARD ERROR -- ");
               fprintf (text_out_gfp,
                    "\n     I can't get memory to
                    accept new message.");
               exit (1);
          }

     /*
     ** Block until a new message is available.
     */
          sender_gu = receive (0,(void *)
               new_message_ap, MAX_MESSAGE_LENGTH);
```

(mailbox.c continued)

```
     /*
     ** Check message ID to see if the message is a
     **   request for any ready messages.
     */
          if (new_message_ap->struct_id_c ==
               ID_REQUEST_MESSAGE)
          {
               post_a_message_vf ();

     /*
     ** If it is not a request for a message, it must
     **   be a message. Add it to the queue.
     */
          } else {
               add_message_to_queue_vf
               (new_message_ap);
          }
     }
     fclose (text_out_gfp);

     return (0);
}
```

```
/*******************************************************
*
*                          ADDMESSG.C
*
*******************************************************/

#include <stddef.h>
#include <systids.h>
#include "mailbox.h"
#include "/clips/clips/include/messages.h"

void add_message_to_queue_vf (link_ap new_message_ap)

{
     extern    unsigned            sender_gu;
     extern    link_ap             list_top_ap;
     extern    link_ap             list_bottom_ap;
     extern    id_only_mat         empty_reply_gma;

     new_message_ap -> next_ap = NULL;
     if (list_top_ap)
     {
          list_bottom_ap->next_ap = new_message_ap;
     } else {
          list_top_ap = new_message_ap;
     }
     list_bottom_ap = new_message_ap;
     reply (sender_gu,&empty_reply_gma,
          sizeof (id_only_mat));

     return;
}
```

```
/*********************************************************
*
*                          REMVMSSG.C
*
*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <systids.h>
#include "mailbox.h"
#include "/clips/clips/include/messages.h"

post_a_message_vf ()

{
     extern    unsigned       sender_gu;
     extern    link_ap        list_top_ap;
     extern    link_ap        list_bottom_ap;
     extern    id_only_mat    empty_reply_gma;
     extern    FILE           *text_out_gfp;

               link_ap        released_mssg_ap;

     if (list_top_ap)
     {
          reply (sender_gu,
               (void *) list_top_ap,
               MAX_MESSAGE_LENGTH);
          released_mssg_ap = list_top_ap;
          list_top_ap = list_top_ap -> next_ap;
          if (! list_top_ap)
          {
               list_bottom_ap = NULL;
          }
          free (released_mssg_ap);
     } else {
          reply (sender_gu,&empty_reply_gma,
               sizeof (id_only_mat));
     }

     return;
}
```

APPENDIX G

Text Output Files

NOTE: Screens have been altered to conform to printing requirements.

The following is a copy of the user screen as seen when the example edits are executed:

```
         CLIPS (V4.10 10/05/87)
CLIPS> (load "rules.c")
*********************************************************
***************
CLIPS> (reset)
CLIPS> (run)
What is the name of the file for the first take?
herw.tim
What is the name of the file for the second take?
herw.tim
What is the name of the file for the output take?
try1.tim
What is the start of the last note before the edit
     from the first take? 44310
What is the start of the first note after the edit
     from the first take? 57950
What is the start of the last note before the edit
     from the second take? 88600
What is the start of the first note after the edit
     from the second take? 96210
Backtracking because of disagreement. Trying for
     frequency.
Agreement has been reached on edit point placement
Agreement has been reached on edit point placement
Do you approve of this edit? (y/n)
y
Am I good or what?
Do you want to do another edit? (y/n)
y
What is the name of the file for the first take?
herw.tim
What is the name of the file for the second take?
herw.tim
What is the name of the file for the output take?
try2.tim
What is the start of the last note before the edit
     from the first take? 363000
What is the start of the first note after the edit
     from the first take? 377000
What is the start of the last note before the edit
     from the second take? 418600
```

(user screen continued)

```
What is the start of the first note after the edit
     from the second take? 423800
Backtracking because of disagreement. Trying for
     frequency.
Agreement has been reached on edit point placement
Agreement has been reached on edit point placement
Do you approve of this edit? (y/n)
y
Am I good or what?
Do you want to do another edit? (y/n)
y
What is the name of the file for the first take?
herw.tim
What is the name of the file for the second take?
herw.tim
What is the name of the file for the output take?
try3.tim
What is the start of the last note before the edit
     from the first take? 363000
What is the start of the first note after the edit
     from the first take? 377000
What is the start of the last note before the edit
     from the second take? 393600
What is the start of the first note after the edit
     from the second take? 408800
Backtracking because of disagreement. Trying for
     frequency.
Agreement has been reached on edit point placement
Agreement has been reached on edit point placement
Edit rejected by envelope expert because
     next-attack-amplitude-decrease
The editor could not accomplish an acceptable edit
     here.
Were you able to complete an acceptable edit(y/n)?n
Some things just were not meant to be.
Do you want to do another edit? (y/n)
n
```

The following is a copy of the debug screen while the three example edits are performed on an Intel 386 33MHz machine without a math co-processor. Note that the examples were done with a math co-processor on a 25MHz machine. This will alter the order in which the tasks are completed and hence the order in which the messages appear. Note also that the screen format has been changed to conform to printing standards.

```
Login:
Starting experts
Envelope setting take to herw.tim.
Envelope got sampling rate of 44100.
Envelope setting take to herw.tim.
Envelope setting range to 44310 57950.
Envelope setting lowest frequency to 50.
Spectrum getting sampling rate.
Spectrum setting take.
Spectrum setting range.
Rhythm setting take to first take.
Rhythm setting range to 44310 57950.
Spectrum searching for an edit.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum found higher lowest frequency = 279.
Spectrum doing FFT analysis.
Envelope expert searching for edit.
Envelope Expert saving maximum crest value of 1473 for
     take 0.
Rhythm setting edit from source 1 to 55903
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum posting lowest frequency.
```

(debug screen continued)

```
Envelope setting lowest frequency to 279.
Envelope expert searching for edit.
Envelope Expert saving maximum crest value of 1473 for
     take 0.
Rhythm setting edit from source 1 to 57529
Spectrum searching for an edit.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum found higher lowest frequency = 322.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Envelope expert searching for edit.
Envelope Expert saving maximum crest value of 1916 for
     take 1.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum found higher lowest frequency = 409.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
```

(debug screen continued)

```
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Decision maker creating edited file.
Decision maker adjusting header information.
Decision maker is generating a crossfade.
Decision maker completed edit attempt.
Envelope evaluating edit.
Rhythm evaluating edit.
Spectrum evaluating edit point
Envelope accepting edit.
Rhythm accepting edit.
Spectrum accepting edit.
Envelope setting take to herw.tim.
Envelope got sampling rate of 44100.
Rhythm setting take to first take.
Spectrum getting sampling rate.
Rhythm setting range to 363000 377000.
Spectrum setting take.
Spectrum setting range.
Envelope setting take to herw.tim.
Envelope setting range to 363000 377000.
Envelope setting lowest frequency to 50.
Spectrum searching for an edit.
Spectrum doing FFT analysis.
Envelope expert searching for edit.
Envelope Expert saving maximum crest value of 2470 for
     take 0.
Spectrum analyzing FFT looking for maximums.
Rhythm setting edit from source 1 to 375625
Spectrum looking for lowest frequency.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
```

(debug screen continued)

```
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum posting lowest frequency.
Envelope setting lowest frequency to 409.
Envelope expert searching for edit.
Envelope Expert saving maximum crest value of 2470 for
     take 0.
Rhythm setting edit from source 1 to 376724
Spectrum setting take.
Spectrum setting range.
Envelope setting take to herw.tim.
Envelope setting range to 418600 423800.
Envelope setting lowest frequency to 50.
Envelope setting lowest frequency to 409.
Rhythm setting take to second take.
Rhythm setting range to 418600 423800.
Spectrum searching for an edit.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum found higher lowest frequency = 473.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Envelope expert searching for edit.
Envelope Expert saving maximum crest value of 2363 for
     take 1.
Decision maker creating edited file.
Decision maker adjusting header information.
```

(debug screen continued)

```
Decision maker is generating a crossfade.
Decision maker completed edit attempt.
Rhythm evaluating edit.
Rhythm accepting edit.
Spectrum evaluating edit point
Envelope evaluating edit.
Spectrum accepting edit.
Envelope accepting edit.
Envelope setting take to herw.tim.
Envelope got sampling rate of 44100.
Rhythm setting take to first take.
Rhythm setting range to 363000 377000.
Spectrum getting sampling rate.
Spectrum setting take.
Spectrum setting range.
Envelope setting take to herw.tim.
Envelope setting range to 363000 377000.
Envelope setting lowest frequency to 50.
Spectrum searching for an edit.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Envelope expert searching for edit.
Envelope Expert saving maximum crest value of 2470 for
     take 0.
Spectrum looking for lowest frequency.
Spectrum doing FFT analysis.
Rhythm setting edit from source 1 to 375625
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum posting lowest frequency.
Envelope setting lowest frequency to 473.
Envelope expert searching for edit.
```

(debug screen continued)

```
Envelope Expert saving maximum crest value of 2470 for
     take 0.
Rhythm setting edit from source 1 to 376724
Rhythm setting take to second take.
Rhythm setting range to 393600 408800.
Envelope setting take to herw.tim.
Envelope setting range to 393600 408800.
Envelope setting lowest frequency to 50.
Envelope setting lowest frequency to 473.
Spectrum setting take.
Spectrum setting range.
Spectrum searching for an edit.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Envelope expert searching for edit.
Envelope Expert saving maximum crest value of 1381 for
     take 1.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Spectrum doing FFT analysis.
Spectrum analyzing FFT looking for maximums.
Spectrum looking for lowest frequency.
Spectrum comparing maximums.
Decision maker creating edited file.
Decision maker adjusting header information.
Decision maker is generating a crossfade.
Decision maker completed edit attempt.
Envelope evaluating edit.
Envelope rejecting edit because of attack amplitude
     decrease.
Rhythm evaluating edit.
```

(debug screen continued)

```
Rhythm accepting edit.
Spectrum evaluating edit point
Spectrum accepting edit.
```

BIBLIOGRAPHY

1. Balaban, Mira "The TTS language for music description" *International Journal of Man-Machine Studies* 28 (1985): 505-523

2. Chafe, Chris and Mont-Reynaud, Bernard and Rush, Loren "Toward an Intelligent Editor of Digital Audio: Recognition of Musical Constructs" *Computer Music Journal* 6 (1982): 30-41

3. Foster, Scott and Schloss, W. Andrew and Rockmore "Toward an Intelligent Editor of Digital Audio: Signal Processing Methods" *Computer Music Journal* 6 (1982): 42-5

4. Giarratano, Joseph and Riley, Gary "Expert Systems" (1989) PWS-KENT Publishing Co.

5. Gordon, John W. "System Architectures for Computer Music" *Computing Surveys* 17-2 (1985): 191-233

6. Hendler, James and Tate, Austin and Drummond, Mark "AI Planning: Systems and Techniques" *AI Magazine* (1990): 61-77

7. Hyperception "Hypersignal User's Manual" (1986) Hyperception, Dallas, TX

8. Kolodner, Janet L. "Extending Problem Solver Capabilities Through Case-Based Inference" *Proceedings of the Fourth Annual International Machine Learning Workshop* (1987)

9. Loy, Gareth and Abbot, Curtis "Programming Languages for Computer Music Synthesis, Performance, and Composition" *Computing Surveys* 17-2 (1985): 235-265

10. Parks, T. W. and Burrus, C. S. "DFT/FFT and Convolution Algorithms Theory and Implementation" (1985) John Wiley and Sons Publishing Inc.

11. Pennycook, Bruce W. "Computer-Music Interfaces: A Survey" *Computing Surveys* 17-2 (1985): 267-289

12. Roads, Curtis "Research in Music and Artificial Intelligence" *Computing Surveys* 17-2 (1985): 163-190

13 Quinn-Curtis "Science/Engineering/Graphics Tools Revision 7.0 Manual" (1990) Quinn-Curtis, Needham, MA