1. INTRODUCTION

An computer music systems have become more complex and sophisticated, there is often a corresponding increase in the level of expertise required to use them effectively. As a result, a musician typically spends a lot of time "learning the system," instead of making music. What is needed is a user interface which a musician can use with little or no formal training yet sufficiently powerful to handle all of his requests. This paper will describe a natural language interface for musical applications which is both easy to use and sophisticated enough to handle complex applications programs. The system described is designed to be tailored to a number of computer music applications ranging from music score editing and MIDI recording to digital editing and reverberation control. In its initial application, it serves as a "front end" to each of these modules, which serve as a more editor and MIDI sequencer.

2. Why Natural Language?

Natural language has several advantages over conventional input techniques:

- **Very easy to learn and use.**
  A system which knows the natural language of a musician requires little or no prior knowledge.
- **Sufficient power for sophisticated users.**
  An user can gain experience with the system, the system grows in sophistication with them.
  Thus, there is a natural progression from beginner to experienced user.
- **Ideal access to a wide range of applications**
  With a fully integrated natural language system, the user of the system from a user's point of view is the same across all applications. A musician need not learn one syntax for MIDI sequencing, another for score editing, another for digital processing of sound, etc.
- **Honest access to tools**
  With a natural language system, users have immediate access to all tools at the same level.
  There is no need to sift through menus or palettes as with some graphic interfaces.

• More natural use of intelligent tools
  As more intelligent tools are developed, they can easily be integrated into the environment.
  The user can then use these tools in the most natural way possible.

• Does not require expensive graphic input devices
  A natural language user interface can run on an inexpensive CRT. This is ideal for low-budget situations like class support.

• Ultimately hands-free operation
  As voice-recognition systems become available, their integration into a natural language system will enable a musician to work with a system without being tied to a standard computer terminal.

3. System Architecture

The natural language system has four main components: a dictionary, an internal database, an augmented transition network parser (ATN) and an encoder. Figure 1 shows the overall system architecture.

3.1. The Dictionary

The dictionary contains all words and word types used by the parser. Words in the dictionary include all the verbs (action words) which the system knows about, "music-related" nouns, adjectives, and other miscellaneous words. If a word is not found in the dictionary, it is classified to see if it belongs to one of a number of word types. Words types are special categories of words which specific examples are either unclassifiable or simply too numerous to include every example in the dictionary. Examples of word types are *number* (ordinal), *object* (any noun of objects which parses as a legal pitch name), *file* (a *data file*), *soundfile* (any non-string soundfile). When the system gets a hard to determine word type (e.g. "file", "soundfile," etc.) the word itself gets inserted into the dictionary for the duration of the session. Thus, future uses of the word are handled much more efficiently.

Attached to each word and type are features used by the ATN. These include:

• Syntactic category
  The syntactic categories for the word (noun, verb, etc.)
Figure 1. System architecture

- Root
  If the word is not in the root form (inclusive for
  verbs, singular for nouns), the root is used for
  syntactic and semantic processing.

- Semantic Class
  Each noun is assigned to one or more classes
  which break the nouns into groups of similar
  types.

- Miscellaneous Syntactic Information
  Each word can have any number of other
  features. These include verb tense, number
  (singular or plural) etc.

- Case Grammar Links
  Each verb is given a link in the case grammar.
  This will be discussed later.

- Semantic Definitions
  Each word has an associated semantic definition
  used by the semantic features to build the
  internal logical representation. This will also be
  discussed later.

Figure 2 shows some sample dictionary entries

3.2. The Internal Database

All of the information at the system's disposal is stored in
an internal associative database. This includes all
instances of nouns the system needs to know about (for
example, instruments, channels, soundfiles, etc.), and any
other information necessary (i.e., time signature markings,
phrase markers, etc.). All information is stored in a
predefined relational representation. For example,

(Instance base-2 base)

Represents the fact that there is an instance of a base,
which is known to the system as base-2. Likewise,

(Instance flute-18 flute)

(Quality flute-18 good)

represents some flute, known to the system as flute-18,
which is good in quality. To access the internal informa-
tion, we use a pattern-matching retrieval function with
pattern-matching variables. For example, the request

(retrieve-val ?x (Instance ?x base))

will return a list of all the instances of "base" in the
database. By using a conjunctive request, a more ad-
justive database retrieval may be done. For instance, if we
need to find "the good flute parts," we make a request of
the database as follows:

(retrieve-val ?x (and (Instance ?x base) (Quality base ?x good)))

This will return a list of only those flutes in the internal
database which have a quality good.

base
category | (noun adjective)
class | singular
num
second
category | (noun cardinal)
class | second
num
transpose
category | (verb)
root | transpose
tense
tenseless

Figure 2. Sample dictionary entries
The database retrieval system is augmented with simple deductive capability. In fact, even if not explicitly in the database will be needed if that item can be inferred, from facts (data plus rules) which are in the database. The deduction is done by logic, which can be done directly from the database, but only if it is possible to add the facts that without explicitly entering the facts into the database. For example, to process a noun phrase like "the brain-pants," we would expect the database query to be something like...

and expect it to return all instances of "brain" in the database, but database, however, has no direct instance of "brain." Instances of instruments are stored as special instruments, like...

...as a result, the above database query would not succeed. To fix this, we add the backward-chaining rule like "if X is an instance of a trumpet, then X is an instance of brass instruments," and so on. We can do this by adding the following rules with the special underscore/precedence predicate "_C_" to the database...

3.3. The Parser

The user's input is passed to an extended transition network parser. The ATN parser is a state network parser which consists of sub-networks. These sub-networks consist of state nodes and syntactic match arcs associated with context-dependent rules and actions, and parsing and acceptance. Figure 3 shows a small context-free graph as a part of the ATN. As each word is pro-...
transpose the bass up a coda.

The case grammar routines would mark this as anomalous by noting that “transpose,” when modified by a prepositional phrase beginning with “up,” requires the noun of the prepositional phrase be of semantic class musical interval. Thus, “up a coda” cannot modify “transpose,” since “coda” is of semantic class “musical motion.”

<table>
<thead>
<tr>
<th>Play</th>
<th>Preposition</th>
<th>Semantic Class</th>
<th>Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>from</td>
<td>musical-section</td>
<td>StartTime</td>
</tr>
<tr>
<td>to</td>
<td>to</td>
<td>musical-section</td>
<td>EndTime</td>
</tr>
<tr>
<td>direct-object</td>
<td>direct-object</td>
<td>instrument</td>
<td>Instrument section</td>
</tr>
<tr>
<td>direct-object</td>
<td>direct-object</td>
<td>musical</td>
<td>StartTime</td>
</tr>
</tbody>
</table>

**Figure 4. Parse tree for “transpose the base up a fifth.”**

### 3.3.2. Noun Phrase Semantics

Consider the sentence, “play the first trumpet part.” Using the case grammar routines, we have determined that the action, play, is to be carried out on the musical instrument described by the noun phrase, “the first trumpet.” However, we do not know how the specific noun phrase translates into an internal representation suitable for use by the applications program. The semantic routines do this by constructing the sense of the noun phrase from the words, and by determining the referent of the noun phrase from the sense. The referent will eventually be bound to internal representation which the applications program will use. Let’s suppose the internal database holds the following information:

- (Instance trumpet-18 1)
- (Instance trumpet-22 2)
- (Number trumpet-18 1)
- (Number trumpet-22 2)

representing two trumpet parts, first and second, which have representations in the applications program, trumpet-18 and trumpet-22 respectively. When the parser is parsing the word “first,” it executes the semantic definition for “first.” This adds the database query

- (Number 1x 1)

for the sense of the noun phrase. When the semantics of “trumpet” is processed, the semantics of “trumpet” is executed, adding

- (Instance 1x trumpet)

for the sense of the noun phrase. At this point, the sense of the noun phrase is

- (and (Instance 1x trumpet) (Number 1x 1))

ICMC 86 Proceedings 122
Finally, the semantics of "the," which had been put on hold when it was parsed, are executed. The semantics of "the" causes the appropriate database query procedure out of the sense of the noun phrase. So, the sense of the word "the" becomes (self (the referent of SNP) (restrict to 1 x and distance 5 front) (Novemb' r 8 13) [where SNP is the noun phrase). Finally, the sense of SNP is extracted, setting the referent to the name returned by etymology+val. In this case, transp-2. 

Just as with the dictionary, there are a number of word classes whose members are too numerous to list individually in the internal database. For example, "second," in "second x," or "half x" are both typical noun phrases. However, it would be impossible to represent every meaningful noun across point in the internal database. To solve this problem, we augment the semantic definition of words whose referents are too likely to be found in the internal database. In addition to the semantic definition, we include the name of the procedure for retrieving the referent. We then change the semantic definition of determiners (i.e., x, etc.) so that if the noun phrase referent is not to be found in the internal database, it uses the procedure specified by the semantics of the noun.

3.3.3. Sentence Semantics

The sentence semantic procedures create the final internal representation of the logic sentence. Each syntactic unit (subject, direct-object, verb, etc.) has its associated semantic definition. After a sentence is syntactically parsed, sentence semantic routines construct the sense of the sentence by assigning the senses of each syntactic unit. Consider the sentence, "play the first piano of the band." The semantic of "play" cure to create a new event, say event-33 whose action is "play," the sense of the sentence then becomes

(Action event-33 play)

"play" is a transitive-action subject of the event, so it must be specified. We use the same general rules to derive the sense of the sentence. Assuming the "first piano" in lines 2 to 6, the sense of the sentence is now (and (Action event-33 play)

(StartTime event-33 34.0)

(EndTime event-33 40.0)"

Likewise, the semantics of "piano" are specified as the sense of the corresponding representation of "piano" to be added to the sense of the sentence. Assume that "piano" is the referent of "piano," the sense now becomes

(and (Action event-33 play)

(StartTime event-33 34.0)

(EndTime event-33 40.0)

(Instrument event-33 34.0))"

Finally, semantic routines check the mood of the sentence as determined by the parser (inceptive or declarative). The semantic definition of each mood describes how the sense of the sentence is to be connected. In the case of declarative sentences, like "the door is open," and "the wall is white," the sense is inserted into the internal database.

3.4. Resolving Syntactic Ambiguity

Consider the sentence, "show me the first trumpet and flute parts in measure 5." From a syntactic point of view, there are several interpretations of this sentence. The prepositional phrase, "in measure 5" can be interpreted as modifying either the verb, "show," the noun phrase, "the first trumpet and flute parts," or ask "first parts." Also, the prepositional "in" may modify either "trumpet" or "flute parts," or ask "trumpet flute." To decide which of these possible syntactic parses represents the user's intention, at the sentence parsed, it is given a likelihood rating. The likelihood rating is an integer representing the system's ranking of plausibility of one intention. A higher score represents a more plausible sense; a positive value means the parser could not infer the sense of the sentence, even though it passed the syntactic parse. A parse's score can be modified in a number of different ways. For example, if any noun phrase has no referent, i.e., there is no item in the database corresponding to the noun phrase, the likelihood rating decreases by a very large amount, and the parse is ranked describing the event. Also, parses which have prepositional phrases which match the case grammar of the verb are given a higher likelihood rating than parses which have no matches. Thus, the parse in which the noun phrase modifies the verb, "show," will be considered more likely than any other.
The user maintains a continuing dialog with the system allowing partial specification of commands where context makes them clear. For example, after the command "play the drum from measure 24," the bulletin board will look something like this:

**Verb**
play
**Command**
Instrument
**Score**
base-35
**Start Time**
06:00

If the next command is "add the drums and guitar," the posting function for "add" is used by the executor to post the results of the parse. For add, the posting function says to insert the "Command" slot of the bulletin board unchanged, and add the other slots where appropriate. This will result in the following bulletin board configuration:

**Verb**
add
**Command**
Instrument
**Score**
(base-35 drums-16 guitar-2)
**Start Time**
09:00

Figure 7. A dialog with the language system using elesed
As a result, when the application function made the bulletin board, it made the command to play the ham, drums and guitar from measures 24.
Each verb the system understands has an associated application function. This function is called by the user when whatever is in the "Command" slot on the bulletin board. It contains all the information necessary to construct the proper command to be sent to the application program from the information on the bulletin board. In the case of score editing, these programs create and command lines and send them to Aldor, which executes them. Figure 7 is a section of a dialog with the natural language system, and the red commands each sentence creates.

6. Support Tools
The system includes a set of support tools to help the system administrator in debugging new applications. The dictionary editor allows the administrator to add words to the system and modify their syntactic and semantic properties. A grammar editor is included to help in the process of verifying the syntactic grammar which the system understands. In addition to the editor, a number of graphical support utilities are included for displaying the system's grammar and for graphing parse trees of individual sentences.

6. Plans for Future Work
At the writing of this paper, the natural language processor has been given English input and created and compiled similar to those generated. Future planned applications include incorporation into a digital music/digitizing environment for NCM, and a similar design system using KISM.

The major problem with the system as it currently exists is its limited knowledge of the musical domain. As a result, user requests must be explicitly stated, using specific terms and commands. For example, a command like "take it from the c chord in the second verse" poses several problems for the system. Firstly, the so-called "verses" would have to be have been defined explicitly by the musician using the system in terms of either sets of measures, or measures we used, he must qualify the absolute time signature(s). Secondly, since the system has limited access to the score on a micro-level and does not have knowledge of harmony, the system will not be able to determine a reference for the "c chord," and all panels would have a negative Sklant rating. What's needed is a way to access a huge base of musical knowledge which can examine the score and return its insights to the system. Current plans are to incorporate the rule-based Knowledge System for Music (KISM) into the natural language processing system. KISM will act as an augment to the internal database of the system. If a reference is made for a music phrase, the system will check to see if the reference can be found using the rules of KISM. If so, control will be handed to the knowledge system to determine the proper reference. KISM, along with sophisticated sets of verbs, will add needed "intelligence" to the system. These rules will include rules on the meaning, such as for chord and time finding, and more complex ones, like planet and meter determination. Figure 8 how KISM and the natural language systemoff-eventually coordinate.

References