A DATA STRUCTURE FOR AN INTERACTIVE MUSIC SYSTEM

by

Stephen W. Smoliar

Technical Report No. 21

October 1972

ABSTRACT

The EUTERPE2 data structure is a representation of a musical score. This data structure is designed so that it may be readily manipulated through an on-line interactive system. The language of this system is an extension of the machine language of the computer on which the data structure is implemented. The current version of EUTERPE2 runs on a PDP-15.
A DATA STRUCTURE FOR AN INTERACTIVE MUSIC SYSTEM

Stephen W. Smoliar

EUTERPE ([Smoliar, 1971a]) is an on-line interactive system for musical composition and style analysis. The initial version was implemented on a PDP-6 and now runs on this machine under the PDP-10 ITS time-sharing system at the MIT Artificial Intelligence Laboratory [Smoliar, 1971b]). This system, while useful, is impractical as it demands an unnecessarily unwieldy computer facility. EUTERPE2, described herein, is an attempt to scale EUTERPE down to a more convenient size. The facility for the current implementation is a [PDP-15] running under DEC's DOS operating system; but EUTERPE2 is, for the most part, independent of operating system details. In fact, we foresee the need to forego the assembly, loading, and debugging utilities as provided by DOS in favor of a more musically-oriented operating system.

The EUTERPE2 data structure, like that of EUTERPE, is a representation of a musical score, where we define the term "score" abstractly, as in [Smoliar, 1972]:

A score is an ordered configuration of symbols called notes which, when evaluated with respect to exterior parameters of pitch and duration, represent an ordered sequence of sounds. Furthermore, simultaneous occurrences of these sounds may be decomposed into the superposition of voices, each of which has no such simultaneous occurrence of sounds.

The interpretation of a score, as defined in these terms, depends upon the functioning of a software-simulated EUTERPE2 "processor" which is organized as follows:
Six programs, each of which represents a voice, are maintained in parallel. Each of these voice programs consists of note word instructions and control instructions. The former pass data to a sound-producing playing unit according to data stored in the pitch parameter, duration parameter, and articulation units. Control instructions maintain the status of these units, as well as the program counter units (thus effecting transfer of control) and other CPU and memory registers. For a more detailed discussion of this type of structure, we refer the reader to [Smoliar, 1971a].

EUTERPE2 instructions are structurally compatible with [PDP-15] machine instructions whose basic format is as follows:

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>I</th>
<th>X</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 3 4 5 6 17</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The operating instruction code (op code) is given by the high-order four bits in an instruction word, the address by the low-order twelve bits, and the use of a single index register in effective address computation is indicated by the X bit. (I denotes indirect addressing which precedes indexing in effective address computation and may only be applied to one level.) The op codes, together with their mnemonics and interpretations, are ([PDP-15]):

- 00 CAL Call monitor
- 04 DAC Deposit accumulator
- 10 JMS Jump to subroutine
- 14 DZM Deposit zero in memory
- 20 LAC Load accumulator
- 24 XOR Exclusive OR
- 30 ADD ADD, 1's complement
- 34 TAD ADD, 2's complement
- 40 XCT Execute
- 44 ISZ Increment and skip if zero
- 50 AND Logical product
- 54 SAD Skip if accumulator different from memory
- 60 JMP Unconditional jump
- 64 EAE Extended Arithmetic Element instruction group
- 70 IOT Input/Output instruction group
- Register control instructions
- Index instructions
- Register transfer instructions
- Operate instructions
Op codes 64, 70, and 74 do not perform memory references; each of these three individual op codes refers to a group of instructions (called microinstructions) an individual element of which is specified by the bit pattern in bits 4 to 17. Execution of CAL is not necessary to EUTERPE2 and is, in fact, forbidden by DOS; furthermore, EUTERPE2 has no need for the Input/Output instruction group. Thus, unless the op code is 00 or 70, an instruction is interpreted as a PDP-15 machine instruction; and if the op code is 70, it may still be interpreted as a machine instruction unless bit 4 is 0. Otherwise, it is processed as a special EUTERPE2 instruction. (By nature of this design, all EUTERPE2 instructions are implicitly microcoded.)

Note word instructions are distinguished by op code 00 in bits 0 to 3. Bits 9 to 17 form a pitch field in which a notated musical pitch is represented by a nine-bit binary number from 1 to 7778. 7778 indicates a rest and 0 represents a repetition of the pitch value of the preceding note. Intonation is an equal-tempered scale of 72 notes to the octave, so a single unit in the pitch field corresponds to a twelfth-tone, hereafter called a microtone as in [Smoliar, 1971a]. Bits 4 to 8 constitute the duration field, each bit of which represents a different note shape as follows:

\[
\begin{align*}
00001 & \quad \text{\textdag} \\
00010 & \quad \text{\textinter} \\
00100 & \quad \text{\texttilde} \\
01000 & \quad \text{\textddag} \\
10000 & \quad \text{\textbullet}
\end{align*}
\]

By principles of microcoding, we may represent a dotted half-note with 01100 and a double dotted half-note with 01110. In fact, there is no reason why we could not represent a triply-dotted half-note (01111) or even such constructs as 10101, which could only be expressed in music notation as:
The fact that we cannot represent a dotted sixteenth-note, or even a thirty-second-note, in this system is no serious problem. Such durations may be expressed by altering the duration parameter, which we discuss below. In fact, it is best to think of the duration field simply as an integer which, when multiplied by the duration parameter, gives the number of "time units" through which that note endures.

The articulation registers are single registers, one for each voice, each of which designates a fraction of the note's written duration which is held silent before the next note is sounded ([Smoliar, 1971a]). The pitch parameter, duration parameter, and program counter units, however, have a more complex structure. EUTERPE represented these units as a set of sequentially allocated stacks ([Smoliar, 1971a]), but EUTERPE2 uses a linked list structure in which each unit is a configuration of words and pointers which we may represent diagramatically as follows:

(In this notation, a single box, represents a single word in memory; and a double box, , represents two successive words. Arrows indicate pointers, and those boxes which do not contain arrows are data words.) The "vertical" substructures in these units are simple stacks in linked representation ([Knuth]):
However, the top-level list has pointers at both ends. This is because it will be necessary to read this list in the direction of its pointers, while structural modifications will take place at the end of the pointer chain. Thus, we shall call the pointer to the beginning of the pointer chain the **structure pointer**, since it indicates the structure as a whole; and we shall call the pointer to the end the **active pointer**, since it indicates the site of active changes to the structure. The stack substructure indicated by the word to which the active pointer is pointing is called the **currently active stack** of the structure.

The data which are stored in these structures vary for each parameter. Each data word in a program counter unit contains an address (in the format of a PDP-15 memory reference instruction) of some location in the EUTERPE2 program. The **current address** of a particular voice is given by the top word of the currently active stack of its program counter unit; i.e., the word indicated by the double arrow:
Each data word in a duration parameter unit is a pointer to a word pair which represents a rational number as an integer fraction:

```
| denominator | numerator |
```

The structure of a data word in a pitch parameter unit is somewhat more complicated:

<table>
<thead>
<tr>
<th>SIGN</th>
<th>MODE</th>
<th>BASE</th>
<th>DISPLACEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This word represents a displacement of the notated pitch in units of either microtones or scale degrees. The displacement number is given in signed-magnitude form; that is, bits 9 to 17 always contain a positive number and bit 0 serves as a sign bit. If bits 5 through 8 are zero, the displacement is in microtones. Otherwise, this field gives a number from 1 to 12 specifying the pitch on which a scale should be built. Bits 1 and 2 specify the type of scale as follows:
0 0  major
0 1  harmonic minor
1 0  melodic minor
1 1  natural minor

The displacement number then gives the number of degrees of displacement along this scale.

While the interpretation of a program counter unit is simply a matter of finding the current address, the parametric units are more complicated. A parameter is, in the abstract sense, a transformation applied to the notated value. By representing a parametric unit as a list of stacks, we may represent a series of transformations, each of which resides at the top of a stack substructure, applied in the order in which they appear on the top-level list.

To illustrate the necessity for such a series of transformations, we may use the example posed in [Smoliar, 1971a]:

The actual "interpretation" of this is:
Thus, for example, the fourth note on the lower staff arises from two transformations. The specification "in A" represents a displacement of eighteen microtones down:

\[
\begin{array}{cccccc}
& \varnothing & \varnothing & 2 & 2_b \\
0 & 1 & 2 & 5 & 8 & 9 & 17 \\
\end{array}
\]

while the instrument specification, "Corno basso", implies an additional octave transposition downward:

\[
\begin{array}{cccccc}
& \varnothing & \varnothing & 1 & 1 & \varnothing_b \\
0 & 1 & 2 & 5 & 8 & 9 & 17 \\
\end{array}
\]

Representing this as a structure, we obtain

A more likely notation of the bass horn part would be the following:

In this case we interpret the fourth note as a pitch which is one scale degree below the tonic played on a B\textsubscript{b} bass horn. We may derive this pitch from the tonic through the following structure:
Where (c) and (d) are the words:

\[
\begin{array}{cccc}
\text{(c)} & - & \emptyset & \emptyset \\
0 & 1 & 2 & 8 & 9 & 17
\end{array}
\]

\[
\begin{array}{cccc}
\text{(d)} & - & \emptyset & \emptyset & \emptyset \\
0 & 1 & 2 & 5 & 8 & 9 & 17
\end{array}
\]

In this structure, transformation (b) represents the fact that the instrument is a bass horn, i.e. the instrument sounds an octave lower than its notated score. Transformation (c) represents the B\textsuperscript{♭} crook – an additional transposition down a whole tone. Finally, (d) indicates lowering one scale degree in the major scale built on the tonic pitch of the current tonality. We may illustrate these transformations as follows:

\[
\text{\includegraphics[width=\textwidth]{transformation_diagram.png}}
\]

Notice that this series of transformations, as we have drawn it, does not correspond strictly with the notation of the B\textsuperscript{♭} bass horn part. The model is, nevertheless, an accurate one. With respect to the horn part, the notated B is directly represented by transformation (d). However, since transformation (c) changes the notated tonality, this note is never really "seen" by the structure in the form of a notated B. All that matters is its position as a scale degree.
We have used the word "tonality" somewhat loosely in this example; and however controversial the term may be, we are obliged to clarify it on a more formal basis. According to [Berry], "Tonality is the system of musical organization around a particular note (the tonal center, or tonic) toward which all functions within the tonal system ultimately point." For our purposes, we would like a pitch transformation to affect not only the interpretation of a pitch field but also the status of this tonic. Formally, we define the tonic as an implicit unit of information which gives that pitch level which is the interpretation of a notated C. A pitch transformation whose base field is zero will cause the tonic to be shifted by the number of microtones given by the signed displacement (modulo 72);10 thus, transformation (a) lowers the tonic a minor third, transformation (b) leaves it unchanged, and transformation (c) lowers it a major second. If the base field is not zero, the tonic is raised by one less than the number of semitones indicated by the base field. In other words, the tonic becomes the tonic of the scale in terms of which the displacement is given. Thus, transformation (d) does not change the tonic, while a base field containing 8 would raise the tonic a major fifth.

Our use of the key signature in the above example is actually a misleading representation of the tonic, since a tonic may be associated with both a major and a minor key, each of which has a different signature. Our notion of tonic is independent of mode and we would do better to write the example as follows:

\[
\begin{align*}
\text{Tonic: C} & \rightarrow \text{Tonic: C} \quad \rightarrow \text{Tonic: B}^b \quad \rightarrow \text{Tonic: B}^b
\end{align*}
\]
Thus far, we have not been very rigorous about the order in which these transformations are applied. However, a trivial example will reveal that these transformations are not always commutative. Let (e) and (f) be the words:

\[
\begin{array}{c|c|c|c|c|c}
    & 0 & 1 & 2 & 5 & 8 \ 9 & 17 \\
\hline
(e) & + & 0 & \# & 1 & 1 & 0 \\
\hline
(f) & + & 3 & 1 & 0 & \# & \#
\end{array}
\]

Suppose the current tonality is C and the note which is being transformed is:

\[
\text{\begin{figure} \centering} \\
\text{\includegraphics[width=0.5\textwidth]{note}} \\
\text{\end{figure}}
\]

Then two possible transformations obtain:

either:

\[
\text{\begin{figure} \centering} \\
\text{\includegraphics[width=0.5\textwidth]{notes}} \\
\text{\end{figure}}
\]

TONIC: C → TONIC: C → TONIC: C

or:

\[
\text{\begin{figure} \centering} \\
\text{\includegraphics[width=0.5\textwidth]{notes}} \\
\text{\end{figure}}
\]

TONIC: C → TONIC: C → TONIC: C
Hence, the ordering applied to the top-level list is critical to the interpretation of the pitch parameter. In the horn example we have handled such information on a first-in first-out basis. First we prescribe the type of instrument (bass horn), next its crook ($B^b$), and finally the actual note being sounded. Our pointers are thus structured so that the transformations are applied in the same order in which they are supplied to the structure.

The interpretation of the duration parameter unit is performed in exactly the same manner as that of the pitch parameter unit. As we have already remarked in our discussion of note words, a duration field is treated as an integer. This integer is then multiplied by the integer fractions indicated by the tops of the stack substructures. The result is the number of cycles through the sound-producing loop in the playing unit for which that note endures. (Clearly, this loop must be processed for an integral number of cycles; yet the result of these multiplications need not be an integer. This problem is discussed in [Smoliar, 1971a]; and the solution described therein has been directly applied to EUTERPE2.)

A structure manipulation operation in EUTERPE2 assumes the following form:

```
    70  VOICES  OPERATION  OPERAND
   0  5  6  11  12  13  15  16  17
```

The voices field specifies which voices will be affected by the instruction. Bit 12 indicates whether the instruction will alter the structure of a parametric unit or simply the top word of the currently active stack. If the structure is to be altered, the operation field gives the type of alterations which will be applied. Finally, the operand field designates which parameter is being affected by the instruction.
Let us now consider these functions in greater detail. The voices field is a generalization of the voice argument in Euterpe ([Smoliar, 1971a]).

Bits 6 through 11 correspond to voices 1 through 6, respectively; and each voice whose corresponding bit is 1 is affected by the instruction. If this field is zero, the only voice which is affected is the one executing the instruction.

Any operation which adds information to a structure must supply an argument. This argument immediately follows the instruction itself, and the flow of control knows how to skip around it. (This is similar to the way in which the argument is provided for the multiplication and division operations in the [PDP-15] Extended Arithmetic Element.) Thus, an argument for the program counter is a full word whose rightmost fourteen bits give an effective address according to the conventions of a PDP-15 memory reference instruction, an argument for the pitch parameter is a full word structured as a data word for the pitch parameter unit, and an argument for the duration parameter is a word pair specifying an integer fraction (first denominator, then numerator). This argument may either replace the top of the currently active stack or it may involve adding a new node to the structure. The former case is indicated by bit 12 of the structure manipulation operation being set to 0.

If, on the other hand, the structure is changed, the effect of the operator is more complicated.

The argument of a change to either the pitch or duration parameter unit may be interpreted in two ways. As is the case when bit 12 is 0, the argument may be interpreted as piece of data which is copied into the structure. We shall call such a change an absolute alteration. Alternatively, the argument may be interpreted as a transformation, which, when composed with the transformation at the top of the currently active stack, yields a new transformation which is the datum which is added to the structure. This change we shall call a relative alteration. (The terminology of "absolute" and "relative" alterations is drawn from [Smoliar, 1971a].)
Likewise, the structure of a unit may be changed in two ways. If we are adding data, it may be pushed onto the top of the currently active stack. Alternatively, we may add a new stack to the list and assign it its first data word. In this latter case, if the unit is the pitch parameter or the duration parameter, the alteration is absolute. Otherwise, it is relative. Of course, if the structure is the program counter unit, we do not regard the argument as a transformation, and all alterations are absolute.

In the case of the pitch parameter, we may consider a relative alteration only if it is compatible with the top of the currently active stack, that is, only if the composition of the two transformations is well-defined. For example, the transformations given by words (c) and (d) are not compatible because the resulting displacement cannot be expressed strictly in terms of either microtones or scale degree. Likewise, the transformations given by words (e) and (f) are incompatible because the compound transformation cannot be expressed in terms of a single mode. Thus, we restrict the data word of the argument of a relative alteration to the displacement field and the sign bit. This signed displacement is added to the signed displacement in the data word at the top of the currently active stack to produce the new transformation, and the mode and base fields are unaltered.

Thus far, we have only considered adding data to a structure. Clearly, we have two corresponding means of deleting data. We may either pop off the top of the currently active stack or delete the currently active stack entirely, in which case the immediately preceding stack substructure becomes the new currently active stack. An addition, EUTERPE2 has auxiliary stacks for saving and restoring data in these structures. Two such stacks are associated with each structure. In one case, the save operation pops a data word off of the currently active stack and pushes it onto an auxiliary stack; and a restore operation pops the data word off of the top of the auxiliary stack and pushes it onto the currently active stack. In the other case, the analogous operation is performed with respect to whole stack substructures. In other words, the save operation places the currently active stack on an auxiliary stack and its predecessor on the top-level list becomes the currently active
stack. Conversely, the restore operation takes the stack substructure at the
top of this auxiliary stack and adds it to the end of the top-level list,
making it the new currently active stack. To summarize, bits 13 through 19
of a structure manipulation operator specify the structure change as follows:

0 0 0 Create a new currently active stack whose initial data is
the absolute argument.

0 0 1 Delete the currently active stack.

0 1 0 Save the currently active stack on the appropriate
auxiliary stack.

0 1 1 Restore the currently active stack from the top of the
appropriate auxiliary stack.

1 0 0 Push a new entry onto the currently active stack according
to the relative transformation given by the argument.

1 0 1 Pop off the top of the currently active stack.

1 1 0 Save the top of the currently active stack on the appropri­
ate auxiliary stack.

1 1 1 Restore the top of the currently active stack from the
appropriate auxiliary stack.

Bits 16 and 17 designate which parameter is being modified:

0 0 articulation
0 1 program counter
1 0 pitch
1 1 duration
Since the articulation data is stored in a register, rather than in a structural unit, we need never be concerned with structure changes for the articulation parameter. Thus, the argument for an articulation change may be placed in bits 12 through 15 of the instruction.

Finally, EUTERPE2 has CANCEL and START instructions similar to those of EUTERPE ([Smoliar, 1971a]). These are indicated by 71 in bits 0 through 5. As in the case for structure manipulation operations, bits 6 through 11 constitute a voice field. A 0 in bit 17 designates a CANCEL instruction while a 1 designates a START. CANCEL deactivates the voices designated by the voice field; and START activates them at an address specified in the word following the instruction itself in the format of a PDP-15 memory reference instruction. This argument is placed in the word at the top of the currently active stack of the program counter unit of any voice affected by the instruction, whether or not this voice is already active. If a voice is active and it is affected by a START instruction, and if it happens to be in the course of sounding a note, this note is interrupted and an immediate transfer of control occurs. On the other hand, changes made by structure manipulation operators while a note is being sounded do not take effect until that note is completed.

This design is still only specified on the "machine language" level. The next step in the development of this system will be the design of an assembler through which these instructions will be expressible in a form convenient to musicians. While scores may currently be effectively realized in EUTERPE2, the language is far from a musical one; and since the assembler is in the "work-in-progress" stage, we would appreciate any comments or suggestions from musicians who would be interested in using the EUTERPE2 system.
Bibliography


