TTrees:
An Active Data Structure for Computer Music

Glendon Diener
Center for Computer Research in Music and Acoustics
Stanford University
GRD%CCRMA-F4@SAIL.STANFORD.EDU

Abstract

TTrees are an active data structure devoted to the hierarchical organization of musical objects into computer music scores. They achieve exceptional flexibility and generality by making no assumptions whatever about the nature of these musical objects–they are, literally, 'pure structure'. As a result, they provide a way of structuring compositions calling for a multiplicity of underlying synthesis technologies.

In this paper, TTrees are presented informally through the development of a simple compositional example. A formal definition of the structure in the context of the mathematical theory of languages is then presented. Two existing implementations supporting very different technologies are offered as evidence of their generality. A discussion of future research possibilities concludes the paper.

Background

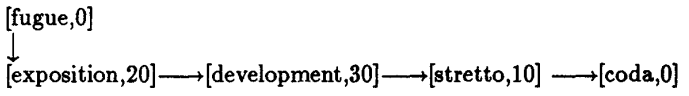
The TTree concept arises from a view of computer music systems as works of art in their own right, independent of whatever their technological merits might be. Designing such systems can be as creatively challenging as music composition itself. A contemporary manifestation of the medieval theoretician's delight in ***musica speculativa***, they can be judged as much for the musical ideas they embody as for the music which they eventually produce. TTrees, in particular, embody in a music composition/synthesis system some of the substance of those hierarchical models of musical form so prevalent in contemporary music theory.

TTrees: An Informal Introduction

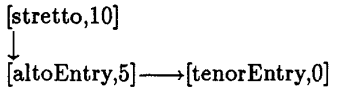
In a TTree, basic musical objects known as SEvents are located in time by a structure of objects called TEvents. These TEvents serve one main purpose: through their hierarchical organization, they structure SEvents into musical scores by specifying their starting times. TEvents maintain 4 state variables:

1. their name.
2. a time value, called 'wait'.
3. a pointer to another TEvent, called 'then'.
4. a pointer to either a TEvent or SEvent, called 'is'.

For convenience, we can describe a TEvent on paper in the form [name,wait], then draw arrows to represent its two pointers: a horizontal one to the right for 'then', and a downward vertical one for 'is'. The following figure shows one way of representing a familiar school fugue form as a TTree:

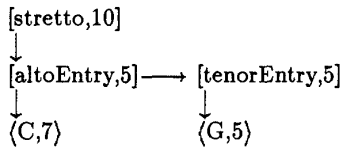


The structure of this fugue can be read directly from the TTree: "fugue is exposition wait 20 then development wait 30 then stretto wait 10 then coda." (Notice that the 'wait' values for the TEvents 'fugue' and 'coda' are redundant). Now let us focus in on the 'stretto' TEvent, making it the root of a new subtree:



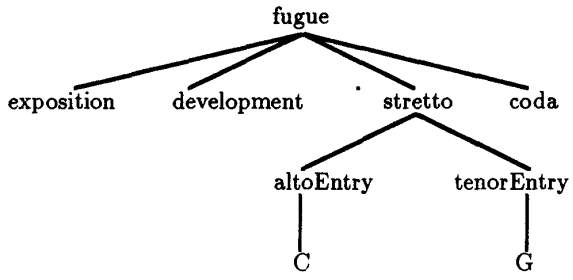
In other words, "stretto is altoEntry wait 5 then tenorEntry". Now, for our 2-part stretto to actually be a stretto, the alto entry must overlap the tenor. This is no problem for the TTree representation, however, provided that whatever 'altoEentry' turns out to be, it lasts for more than 5 time units. The crucial point here is that for any TEvent, its 'wait' value has no relation whatever to its duration. Rather, it specifies a time delay until the beginning of the TEvent designated by its 'then' pointer. TEvents say nothing about duration. For all we know, TEvent 'altoEntry' could last anywhere from nanoseconds to millennia.

So far, this TTree fugue is pure structure—we have said nothing about what it might actually sound like. This is the job of SEvent objects. An SEvent is any low-level, atomic structure that makes sense for a particular sound-synthesis technology. Emphatically, this definition carries no implication whatever that an SEvent is a musical note, at least not in any conventional sense. Nevertheless, to keep this discussion in the realm of the familiar, let us imagine that SEvents are notes, just as they used to be: discrete musical events having definite pitch and duration. On paper, such SEvents can be represented as ⟨pitch, duration⟩ , where the use of angled brackets makes the distinction between TEvents and SEvents immediately clear. To complete the stretto subtree, we compose altoEntry and tenorEntry as follows:

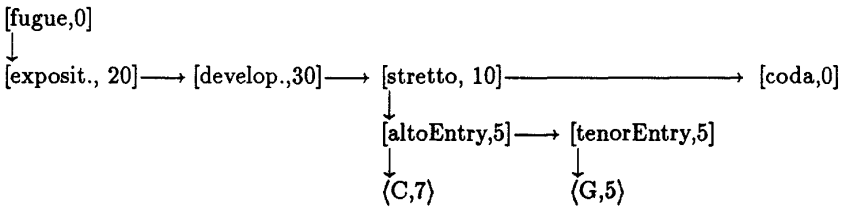


Since 'altoEntry' and 'tenorEntry' overlap by two time units, this is a true stretto. Though admittedly no contrapuntal masterpiece, the example clearly illustrates the difference between a TEvent's 'wait' value and the duration of an SEvent, which is the SEvent's own private property. This distinction is crucial to the flexibility of the TTree concept. SEvents can be anything we wish: a call to a random procedure, an instruction to play a sound file from disk, a message to a robot-conductor instructing it to cue in the first violins. TEvents provide the structure, you fill in the technology.

Two other unique properties of TTrees are worth pointing out. First, note that the TTree is simply a binary tree 'in hiding': it has been rotated counterclockwise a quarter-turn from the way it is normally presented. This rotation is important, however, and is no accident—it preserves the conceptual habits of vertical simultaneity and horizontal succession so deeply engrained in musicians conditioned to common musical notation. Second, the TTree differs from most tree-structured representations of musical scores by placing time along the branches of the tree rather than between them. To illustrate, compare the following 'normal' tree structure:



-with the corresponding TTree:



In the TTree, temporal meaning has been made explicit: to decide when an event begins in relation to any other event, simply add up the 'wait' values along the path between the two events, remembering that vertical branches are not delayed at all. The pitch G in the final stretto, for example, starts 20 + 30 + 5 = 55 time units after the beginning of the score.

T-Trees: A Formal Definition

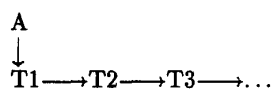
In this section, TTrees are defined in the context of formal language theory. The reader unfamiliar with the theory may safely skip this material without loss of continuity. The formalism adopted here is taken from (Hopcroft and Ullman, 1973).

The idea of describing a data structure for computer music in the form of a grammar was first proposed in (Buxton et al, 1977). A T grammar is a restricted context-free grammar of the following form:

T = (TE,SE,P,S)
-where:
TE is a set of TEvents,
SE is a set of SEvents,
P is a set of productions of the form
$A \rightarrow T^+$, $A,T \in TE$
or $A \rightarrow s$, $s \in SE$,
$S \in TE$ is the grammar's start symbol.

TEvents are the non-terminals of this grammar. Associated with each TEvent is a numeric-valued variable 'wait'. SEvents are the grammar's terminal symbols. Productions in the grammar are restricted to a special normal form. Similar to Chomsky normal, the right hand sides of all productions in T grammars must contain either a single terminal SEvent or an arbitrary number of non-terminal TEvents (in Chomsky normal, no more than 2 non-terminals may appear on the right side of a production).

A TTree is a tree corresponding to a derivation in a T grammar. This correspondence, however, is different than that which exists between a derivation and a tree as conventionally understood in formal language theory. TTrees are strictly binary, and vertical and horizontal arcs replace of the usual left-child and right-child arcs of most binary tree representations. The application of a T grammar production of the form $A \Rightarrow T^+$ will have a vertical arc from A to T1, T1 a horizontal arc to T2, T2 a horizontal arc to T3, and so on. Thus:



The designed purpose of the grammar is to situate SEvents in time. If the root of a TTree has time T, then the time of any SEvent s is the sum of all the 'wait' values associated with all the TEvents having horizontal branches on the path from S (the start symbol) to s.

Implementations

The original TTree-based system was developed on the Synclavier II synthesizer in the electronic music studios of McGill University's Faculty of Music (Diener, 1985). This system, dubbed 'A Music Automata' (AMA), provides the composer with a set of structured editors for creating and editing TEvents and SEvents. The TEvent editor, for example, furnished the capability of editing the 'name' and 'wait' fields of individual TEvent objects. Once a set of such objects is defined, TTrees are created through a command language which mimics the production rules of context-free grammars. These commands effectively promote the TTree concept from a data structure to a language. To illustrate, our fugue example would be created by entering commands similar to the following:

```
fugue ⇒ exposition development stretto coda
stretto ⇒ altoEntry tenorEntry
altoEntry⇒ C
tenorEntry⇒ G
```

A graphic representation of the TTree is drawn on the computer screen as these commands are typed. Acoustic feedback is immediate—either the entire tree or any of its subtrees can be performed in real time at any point in the process.

In AMA, SEvents take the form of recipes for the Synclavier II's digital oscillators. These oscillators have a fixed hardware configuration which AMA models directly, providing slots in the SEvent's structure for each of the oscillator's input ports. AMA, then, rather than abstracting away the oscillator's characteristics, elevates them to instruments in their own right, offering the composer direct control over every aspect of their functionality.

Because it is programmed in a language lacking specific support for object-oriented programming (Scientific XPL), modification of the AMA system requires a fairly thorough knowledge of its internal implementation details. A more flexible approach implements the TTree concept in an environment specifically designed for object-oriented programming. To this end, a set of Smalltalk-80 classes (Goldberg and Robson, 1980), dubbed 'Forest', has been implemented at Stanford University's Center for Computer Research in Music and Acoustics.

In Forest, the 'wait' fields of TEvent objects are themselves slots for arbitrary objects capable of answering 'true' or 'false' in response to the message 'tick'. Whenever a TEvent receives the message 'tick', it immediately relays the message to its own 'is' object. The message is only relayed to its 'then' object, however, if its 'wait' object responds to 'tick' with 'false'. The instance method for the message 'tick' in class TEvent, then, is simply:

```
tick
  is tick.
  (wait tick) ifFalse: [then tick]
```

TTree performance is accomplished by repeatedly sending the 'tick' message to the root TEvent. The instance method for 'tick', aided by its 'wait' object, provides explicit control of the propagation of the message through the tree. The simplist form of 'wait' object, called a 'timer', responds to 'tick' by decrementing an internal counter, answering 'true' if the counter is greater than 0, 'false' otherwise. Other 'wait' objects can be easily created by the composer: one example senses input from the keyboard or mouse, allowing real-time intervention during performance of the tree.

SEvent objects in Forest can be anything which understands the 'tick' message. At present, one such object writes arbitrary midi data to an output port—another computes samples by Fourier synthesis and appends them to a sound file. By keeping SEvent protocol as simple as possible, Forest makes it easy for composers to extend the TTree concept to embrace virtually any synthesis technology at their disposal.

Future Directions

As described in (Buxton et al, 1977), tree-structured score representation schemes, through their implicit inheritance properties, are capable of organizing much more than just the temporal structure of SEvents. Though not formally part of the structure, organization beyond the temporal realm is a natural extension of the TTree concept.

Although Forest provides a command language for TTree specification similar to that of AMA, the power of the Smalltalk-80 environment makes a completely visual programming language interface an enticing possibility. Currently under development, the projected interface will provide a 'cut and paste'-style TTree editor together with a graphically-based tool for the creation and modification of SEvents.

Acknowledgments

Thanks are due to JC and BMR at CCRMA for their interest in this project, to Bo Alphonce and Paul Peterson of McGill University for their contributions to AMA, and to the Social Studies and Humanities Research Council of Canada for their support of my ongoing research at Stanford.

References

Buxton, William, William Reeves, Ronald Baecker, and Leslie Mezei. 1977. "The Use of Hierarchy and Instance in a Data Structure for Computer Music". *Computer Music Journal* Vol.II No. 4.

Diener, Glendon. 1985. *Formal Languages in Music Theory.* Master's thesis, McGill University, Montreal.

Goldberg, Adele and David Robson. *Smalltalk-80: The Language and its Implementation.* Palo Alto: Addison Wesley.

Hopcroft, John E. and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation.* Reading Mass.: Addison-Wesley.