How to Feed Musical Gestures into Compositions

Andreas Mahling
Institut für Informatik
Breitwiesenstr. 20/22, 7000 Stuttgart 80
Germany

Abstract

In this paper we report about the current state of COMPASS, a Smalltalk-80 based system for computer aided composition. We introduce several graphical editors for phrase structure editing and outline an approach that uses the computer for learning from musical examples. The result of the learning process (musical patterns) can be reused for representing phrase structure manipulations and relationships. The musical pattern itself can be the output of a constant matching process that has been applied to a group of examples supplied by the musician. The pattern also can be derived incrementally from a single example, if the musician guides the system by interactively performing abstractions on selected attributes of the phrase example. The set of abstractions and attributes presented by COMPASS is extensible and depends on the entities comprising the phrase example.

1 Introduction

Within the COMPASS Project we have implemented a set of object oriented tools for MIDI-sequencing, for recording and playing musical arrangements, for event list and score editing, etc., to support computer aided composition [Böcker et al. 9]. The most important aspect of COMPASS is its object oriented musical knowledge base. Musical tasks can be supported much better by a system which has access to such a knowledge base. Without it, it is sometimes nearly impossible to adequately support even some of the more pragmatical musical tasks like the correct recognition and transcription of modulations in modal music.

One of the new tools of COMPASS is a system for phrase structure editing musical compositions. It consists of several editors, each of which is supposed to be used for manipulating the structure of a composition at a different level of abstraction. According to their application domain the editors can be split into two groups. Editors of the first group are better suited for working with the time independent relationships existing within a composition whereas editors of the second group are intended for specifying the composition's musical form.

A composition consists of a set of objects each of which is supported by concepts of the musical knowledge base. Editors of the first group relate the entities of the composition to one another by making them the inputs and outputs of (software) processors which manipulate them. Processors can be used for analyzing or producing music or both. They are able to communicate any type of object, not just numbers. This characteristic feature guarantees that, as an extreme example, the same set of processors may be used either to produce MIDI data or lyrics. The order in which processors will be triggered when evaluating a phrase structure net is independent from the time relationships between entities of the composition, e.g., its musical form or arrangement. The musical form can be further edited with editors of the second group.
Musical material, e.g., a set of durations, can be specified by means of score editors, a text representation or musical gestures. In this context, a musical gesture stands for a phrase pattern, i.e., a phrase which is not fully specified. Adding a musical gesture to the system starta by playing a set of similar phrases and have the system perform a kind of constraint matching, i.e., abstraction among these patterns. The abstraction process can be controlled interactively. Our teaching experience at the music conservatory shows that this way high-level musical concepts can be easily defined by a musician. Closing a loop, the result of the abstraction process, the phrase pattern, can be reused in both groups of editors.

2 The Musical Knowledge Base

According to the different abstraction levels, COMPASS divides its knowledge base into three major parts. The first and lowest level consists of MIDI messages, events and other MIDI related objects. It’s not superfluous to represent even MIDI messages as objects, because implemented this way they could easily be adopted to extensions of the MIDI message protocol, e.g., the MIDI Time Code or MIDI Sample Dano Standard. Furthermore, the MIDI message classes could be used to cover environmental information e.g., how many parameters a message has and what kind of the parameter values are. Especially these informations are predispositioned to be located in a single memorable object, because they will have to be used frequently in a system for computer aided composition.

![Diagram of entities and relationships of tones](image)

Figure 1: Excerpt of the entities and relationships of tones.

The top level of the knowledge base is composed of objects that reflect the taxonomical aspect of musical knowledge. It also contains objects that handle different and possibly incommensurable or contradictory interpretations of a musical object or an entire composition. One of the design requirements for the top level of the knowledge base was to be able to include as many (musical) aspects as possible. Fig. 1 gives an informal insight into some of the objects of the musical knowledge base and their relationships to which we also refer by the term deep musical knowledge. Tones, pitches, intervals, scales and chords are examples of objects affiliated with the taxonomical part. contexts are the basic objects for representing interpretations.

A PlayContext keeps all the information that is needed to have a composition played by COMPASS, i.e., by the computer. PlayContexts also provide a kind of device independency in that they help isolating device or system dependent information like the resolution of event time stamps and durations from the music. They also reduce the amount of redundancy in representations of musical compositions: PlayContexts are organized in a hierarchical inheritance lattice. A parameter value common to each tone of a melody needs not to be copied to each tone of that melody but can be inherited from a PlayContext surrounding the melody (see fig. 1). A HarmonicFunctionInterpretation object is yet another example. It accounts for the variety of harmonic functions a chord can be associated with, depending on the context.
We also designed a layer that is capable of interpreting MIDI data in several ways. Drum computers, for instance, encode their instruments via the key number parameter of MIDI note messages. Mixing console fader automations often use the key number parameter of MIDI note messages for encoding the channel number and the key velocity parameter for encoding the fader position, i.e., for adjusting loudness. I.e., depending on the context a MIDI message may get different interpretations.

The intermediate level of the knowledge base is a compromise between the deep musical knowledge and the more technically motivated MIDI standard. MIDI on its own is not enough because it doesn’t integrate higher level musical concepts, e.g., the concept of a cadence or a harmonic function. A musician that nevertheless wants to work with such higher level concepts is forced to translate them to MIDI himself if he uses a MIDI representation only. On the other side, heavily working with objects of the ‘deep musical knowledge’ is likely to eat up all computing resources.

To fill the gap between these knowledge base domains we introduced virtual MIDI events. They implement concepts of average complexity and can be used at an reasonable price concerning computing power. E.g., the MIDI standard does not support MIDI note events, i.e., events that have a channel number, key number, attack/release velocity and a duration. A MIDI note event has to be translated into messages of the real MIDI world, i.e., a MIDI note-on/note-off message pair. A more elaborate example is a MIDI-trill, a virtual event that represents a trill and translates itself into a couple of MIDI note events. The MIDI event list editor in fig. 2 works on an event list that contains rests, yet another type of virtual MIDI event.

3 Composing with Musical Phrases

Currently, ComPArs' most powerful tool for composing is the Phrase Structure Editor. However, phrase structure editing is supported by a lot of other graphical editors, e.g., editors for conventional music notation, sequencing, MIDI event lists and attribute sheets. Most of these editors can be spawned from the Phrase Structure Editor on demand. Patchwork [Dalbrie/Duttenhe 90] and Flavors [Hendry 84] are examples of foreign systems related in some sense to ComPArs' Phrase Structure Editor. Fig. 2 shows a snapshot of a typical phrase structure editing session.

The main purpose of the Phrase Structure Editor is to manipulate musical material. The material can be the result of a former sequencing task or can be defined within the scope of the editor itself, e.g., by means of a graphical editor (see the middle left window of fig. 2). The results of the phrase manipulations could be converted to patterns and returned to the sequencer.

A typical Phrase Structure Editor document consists of nets of boxes, spread among the pages of the document. Each box may have named inputs and/or outputs and a label. Inputs or outputs of boxes may be connected to outputs/inputs of other boxes. Boxes visualize either data or procedures. The small icon in the top left corner of each box signals its concrete function or its membership to a function group.

The top window in fig. 2 shows an example of a phrase structure document page consisting of two nets. The top net rhythmizes a chord progression (chords) according to a set of tones and rests (RhythmSet). A fractal generator (Fractal) produces the 8-measure phrase which rhythmizes (Rhythmizer) the chord progression. The result of the rhythmization is stored in a container for global phrases (chords R).

1 Using MIDI note events instead of MIDI note-on/note-off events prevents the user from separating note-on from note-off events. A separation, e.g., due to a split pattern operation, necessarily leads to keeping rests at the end of a pattern or composition.

2 The ComPArs sequence can read and write MIDI Standard File Format of type 0, 1 or 2. It supplies a virtually unlimited number of tracks and the usual sequencing operations like quantizing, transposing.

ICMC 260
The Phrase Structure Editor distinguishes between global and local data structures. Global data structures can be accessed from outside the Phrase Structure Editor whereas local data structures cannot. The chords, chords I, melody II, and chords II boxes in the top window of fig. 2 for example keep global phrases.

The Phrase Structure Editor shares global phrases with associated editors. E.g., spawning a sequencer from a Phrase Structure Editor associates the Phrase Structure Editor with the sequencer. Then, each global phrase created by the Phrase Structure Editor becomes a new entry in the pattern list of the associated sequencer and each pattern recorded with the sequencer can be referenced through a corresponding (global) phrase from within the Phrase Structure Editor.

The Phrase Structure Editor allows (theoretically) unlimited number of definition levels. A box is called to be at definition level n (or a primitive box or a level-0-box), if it is not decomposable. A net is called to be at definition level n, if n is the maximum of the definition levels of its boxes. If a set of boxes is collapsed into a single new box all edges inputs/outputs of the boxes of the net, i.e., inputs/outputs that have not already been connected, become inputs/outputs of the new box.

Each level-0-box that keeps a local data structure, e.g., a musical phrase, cannot be referenced by name from outside the new level-1-box. Boxes for global data structures can be referenced from any point in the Phrase Structure Editor document or from any associated editor. This feature reduces the average number of connections between boxes of a net, because the user can reference a data structure by simply adding a copy of the container of the data structure, i.e., a box, instead of adding a new connection. It also helps in splitting a large net into smaller ones (each of which may be placed on different pages of the phrase structure document) facilitating a much cleaner layout. E.g., in the top window of fig. 2, the user might have 7 chords I boxes each referencing the same
Most of the box types supplied by the current implementation of the Phrase Structure Editor communicate musical phrases. A melody, a chord progression or even an entire musical composition can be a part of a musical phrase. In COMPASS a musical phrase knows much more than that it is just a melody, for example. It also keeps context information and provides a variety of visualizations for graphical editing, e.g., the musician can open an editor for conventional music notation or MIDI event list editing on each musical phrase. Musical phrases have been implemented using the PlayContext objects mentioned in section 2.

The bottom half of fig. 2 shows some examples of graphical editors that can be opened on a musical phrase. The middle-left window has been opened on a phrase that is to be transmitted from the RhythmStream to the Fractal box via its 'out' output. The window in the bottom left corner displays an excerpt of the result of rhythmizing the chord progression and the bottom right window is a MidiEventList Editor that has been opened on the output of the fractal generator. It is possible to open these windows, because the Phrase Structure Editor allows each input/output of a box to be inspected by the user. This characteristic makes the genesis of the musical phrase resulting from evaluating a phrase structure net transparent.

Each of these boxes has a set of attributes that constrain the way the box works. The window in the center of fig. 2 shows a sheet2 article that has been opened on the attribute set of a fractal generator. The Curve Manager at the right of the sheet indicates how the value of $b$ will vary during the generation of the phrase. $b$ is a variable of the equation $x_{n+1} = 4x_n(1 - x_n)$ defining the fractal generator. The length of the phrase to be emitted by the fractal generator is determined at the left of the attribute sheet. Phrase Lengths can be specified in measures, elements or replications of element units.

![Figure 3: A phrase arrangement editor window.](image)

Like the Phrase Structure Editor, the Phrase Arrangement Editor is for outlining the structure of the musical phrases of a composition. The Phrase Arrangement Editor is for organizing these phrases into a lattice that makes up the form of the musical piece. Fig. 3 shows how the phrases produced by the Phrase Structure Editor of fig. 2 could have been arranged with the Phrase Arrangement Editor.

The Phrase Arrangement Editor has access to all global phrases of a set of associated editors, e.g., all patterns of an associated sequencer. The selected global phrase melody $I$ in fig. 3 for example is a prerecorded pattern of the sequencer which doesn’t appear in the net layouts of the Phrase Structure Editor of fig. 2.

---

2 Of course, the data structure may undergo several changes during the evaluation of the phrase structure net.

4 The attribute sheets for each box have been built with the Visualization Construction Kit (VICK), a commercial development tool for building graphical user interfaces. An upcoming article in the Journal for Object Oriented Programming will describe the system in detail.
The left side of fig. 5 shows a very small voice leading example including a transition. The right side of fig. 5 exemplifies how a single voiced input phrase (a) may be extended by a second voice (b) with the aid of the given example. Remember that (b) could have been produced by a box of the Phrase Structure Editor from input (a). The definition of the box itself could have been the result of the learning process to be explained below. The simplicity of the example doesn’t come from restricted capabilities of our approach but facilitates the demonstration of the basic ideas underlying our concept.

The central idea is to derive a pattern from a single musical example or a set of musical examples. Then an arbitrary musical phrase can be matched against this pattern. If the matching process is successful the musical phrase can be undergo several changes according to the musical example(s) the pattern has been derived from. Because the pattern will capture attributes strongly related to the (musical) meaning of the example(s) we will refer to it as a musical gesture. In this sense a musical phrase itself becomes a musical gesture: it simply matches against exactly one pattern.

A pattern can be derived from a set of musical examples by applying a procedure which is called constant matching. The constants matching process compares two musical phrases and replaces the entities or attributes which distinguish the phrases with variables. The resulting pattern then matches against each of the two initial musical phrases. Therefore it is guaranteed that a pattern that has been derived from a set of musical phrases successfully matches against each of the phrases in the set.

However it can become an awkward task to supply as many examples as are needed to derive a pattern that correctly covers the characteristics of the musical gesture the musician has in mind. In this case a better way would be to guide the pattern derivation process of the system by incrementally and interactively perforizing abstractions or selected attributes of a single phrase example.

Fig. 6 visualizes the phases of such a process as is applied to the gesture example of fig. 5. Each phase concentrates on certain aspects of the gesture example and abstracts from particular relationships or entities. The accumulation of the four phases altogether then describes the resulting pattern.

The bordered window at the top left position of fig. 6 represents the starting point of the abstraction process. The user simply has been confronted with the basic entities of the example phrase, the tones. From this point, another entity can be reached by selecting it from a menu that covers all tone-related entities. Each entity itself is able to pop up a menu of all entities that are reachable from it. (See fig. 1 for an excerpt of all entities that can be reached from a single tone.) The same holds for a group of entities for reaching relationship entities.

Perhaps the user may want to start the abstraction process by concentrating on the interval relationships between the pitches of the tones. For that purpose he first selects all five tones and then selects the Distronic Distance relationship entity from the tone-group-entity-menu. The new entities will then be computed and integrated into the visualization. The first generalization step might be to replace concrete pitches in the example phrase with unnamed variables (?) (see top
Building a phrase arrangement is much like building a phrase structure with the Phrase Structure Editor. The musician places boxes representing global phrases on a page of the phrase arrangement document. He then establishes relations between the boxes which fix the succession of the phrases. The relations will be visualized by arcs. E.g., the arc labeled \textit{step} \rightarrow \textit{start from phrase melody I to phrase melody II} in fig. 3 expresses that \textit{melody II} starts immediately after \textit{melody I} has stopped. The arc from \textit{melody I} to \textit{chords I} denotes that \textit{melody I} starts at the same time as \textit{chords I}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{sequencer.png}
\caption{A sequencer window.}
\end{figure}

If a sequencer, a Phrase Arrangement Editor and a Phrase Structure Editor have been associated with each other, the evaluation of a phrase structure net in the Phrase Structure Editor automatically leads to a new track sheet in the sequencer. Fig. 4 presents the sequencer track sheet resulting from the evaluation of the phrase structure net shown in fig. 2 along with the phrase arrangement of fig. 3.

4 From Musical Phrases to Musical Gestures and Back

Although the current implementation of the Phrase Structure Editor provides a multitude of primitive boxes for processing phrases, a composer may want to create his own or modify already existing box definitions. Unlike many other systems, he then will be faced with the implementational details of the system, especially the programming language and/or development environment that was used for building the program.

Even though box definitions can be expressed in COMPASS on a level much higher than that provided by systems that for example don't have access to a musical knowledge/data base, it might be nevertheless very difficult and tedious for a musician to transform her/his visions into programming code. Hence, an approach allowing the musician to formulate box definitions without any programming effort seemed to be more appropriate.

Based on our experience with critical and tutorial systems for learning musical styles [Mack, 1991] we developed a concept that enables a computer to learn box definitions from musical examples. Among others, \textit{voice leading} appears to be one of the adequate topics for verifying our ideas. Voice leading highly depends on the chosen musical style and despite the fact of already existing voice leading heuristics which could be encoded easily, it’s very difficult to program box definitions that produce musically satisfying voice leadings. The attempt of extending or modifying existing (function) definitions often ends up in a dilemma, because the implementor generally cannot decide where to invent modularity. Albeit, the problem domain is incomplete. At any point of time a new example presenting the or another solution to a voice leading problem may be found, e.g., in the musical literature.

ICMC 264
right entity net of fig. 6). This leads to a pattern that recognizes the phrase example (or parts of it) independently from the actual scale degree, which it occurs.

The musician might move from concrete durations in the example phrase to duration relationships (bottom right entity net of fig. 6), and fix that all pitches should be proper to a major scale with an arbitrary key (mid-left entity net of fig. 6). The bottom left entity net just visualizes the emphasis entity for each tone of the example phrase.

5 Acknowledgements

The work reported in this paper was partly made possible by grants of the Deutsche Forschungsgemeinschaft (DFG). The author also wishes to thank Heinz-Dieter Böcker and Ralf Geisselhäuser for their continuous support and encouragement.

References


