Music Space:
A Metaphor for Music Representation and Music Generation.
Ioannis Zanos
Staatliches Institut für Musikforschung, PK
Tiergartenstr. 1, 10785 Berlin, Germany

Abstract
The music space metaphor is a general framework for modeling principles of organization that underlie different musical styles. Musical structure and behaviour is defined in terms of communicating objects of three kinds: spaces, shapes and points. This model presents several advantages from the computational or programming language point of view and from the musical cognitive point of view. Encapsulation and a high degree of abstraction are achieved. Functions and graphs can be mixed indiscriminately in the same system. Complex structures of musical material can be implemented efficiently. Non-temporal and highly flexible changes, independent detuning, change of tempo, or "rubato" is easily realized by cascading translations in hierarchically interdependent spaces. Also, the music space model was developed with view to implementing spatial models from cognitive science, which represent the perceived relationships of tones. By means of a "musical dynamics" one can guide the development of a composition by chosen perceptual parameters. A prototype implementation of the model in MAX is presented.

Keywords: Artificial Intelligence in Music, Composition Systems and Techniques, Music Data Structures and Representations, Music Languages, Object Oriented Programming

1 Introduction
Contemporary music just as many non-western musical traditions often deals with sound structures that are difficult to represent with the classical western theoretical apparatus. Computer representations based on this apparatus face the same difficulties as traditional notation. The "common notation" score or "piano roll" metaphors that form the basis for most music editors or sequencer-type tools show severe limitations, which stem from the paradigms underlying the representation: The "piano roll" metaphor, which is also the basis for MIDI, uses the premise of one constant pitch, amplitude and timbre per note, and twelve pitches per octave. More refined adjustments have to be made by additional "modulator signals", which are not congruent with the primary "note" signals (channel based, using different units; see Lerdos and Moore 1985). Similarly, in the time domain, a binary subdivision scheme prevails, making fluctuations of tempo and irregular or compound subdivision schemes are difficult to realize.

The problems arising from these limitations on the one hand, and the need to adapt computer programming techniques to music on the other, have occupied many researchers (Dannenberg et al. 1986, Anderson and Kuivila 1989). One approach which has yielded promising results, is propagating parameters through multiple layers of functions (Desain and Honing 1992). Complex shapes can be thus obtained from the combination of simple functions. In a similar way, Maxim and Zahorija (1994) use a hierarchical layering of performance fields to generate tempo variations that simulate expressive human performance.

The "Music Space" metaphor takes the function combination approach one step further. It is a general framework for integrating different types of musical computational models in one system. It does this by establishing a common communication protocol which hides the differences in the data structure and implementation of the models. This idea stems from object oriented programming and is similar to the Model-View-Controller paradigm (Krasner and Pape 1988). The protocol is based on the paradigm of movement in a space. While the basic types of elements and their manner of communication are well defined, the dimensions and structure of a music space as well as its semantics are left entirely free. The points in a space can represent any parameter such as pitch, duration, intensity as well as their combinations. To represent a piece of music, one may use only one space with complex points, or several interacting spaces, where each space represents a different musical parameter. The level of the representation can vary from low-level, using a small vocabulary of rudimentary operations, to high-level, using a richer vocabulary of specialized operations.
2 Constituents of Music Space

Three kinds of elements make up the Music Space model:

2.1 Spaces

Spaces are objects representing the relationships of musical elements such as pitch, duration, timbre, but also abstract or composite elements, independently of a temporal ordering. A simple example of a space is a one-dimensional array of pitches representing a scale. For representing scales with flexible degrees such as present in modal or the ragga systems it is possible to use more complex structures which split up the scale in subspaces and which represent alterations through higher dimensions. Space can be represented by functions that return the value of a "point" in the space given a set of parameters - the coordinates of the point. This makes it possible to represent spaces with an infinite number of elements. To qualify as a space, an object has to provide two ways to access its elements (points in space):

- Absolute reference. A function mapping a set of "coordinates" to the elements of the space. To each coordinate in the set corresponds a point in the space, which is obtained by sending the space object the coordinate.
- Relative reference. At least one way must be defined for reaching a point in the space given the position of another point and a specification of its relative position to that one. For example, given one point in the space, obtain the next point, previous point etc. This can be described as establishing an ordering of the elements of the space (which may be multidimensional).

2.2 Shapes

A shape is a set of relationships between points of a space. The shape is not bound to a particular position on the space, but can be applied to a space at different positions. In this sense, a shape is a space which has the additional property of application on another space at variable positions. An application on a different position returns a different set of elements. The relationships between the elements of each set obtained by the same shape are constant. (This is comparable to a geometric shape or set of points that can be "translated" to different positions in space). Simple examples of shapes are modes, chords, rhythmic patterns, or pitch class set types. A shape whose elements are ordered sequentially is called a "path". Applying a path onto a space returns a sequence of elements, for example, a melody.

A shape is treated as a space when a point moves in it in the same manner as in a space. Spaces can be cascaded or embedded within each other. For example, apreggiating a trill can be realized by looping through the points of a trill shape (1, 3, 5), which is embedded in the shape of a dactylic accent (1, 3, 5, 6, 9, 10, 12), which is embedded in the space of the tempered chromatic scale.

2.3 Moving Points

The moving point is an object comparable to the "turtle" in LOGO. It can be made to move in a space by sending it messages. As a result it may produce tones or interact with other points. A point "knows" its position in one or several spaces, as well as some additional information that determines its reaction to messages, such as its current orientation in the space, its relationship to other points, etc. Simple kinds of points just return a parameter such as frequency of duration according to their position. Compositions of different point types are also considered points. An "impulse" is the voice module described below, which handles all parameters necessary for the performance of a melodic line.

2.4 Configuration Example

Figure 1 shows a theoretical configuration involving two separate paths, a pitch path describing the pitch structure of a part and a time path describing its temporal structure. Note that the separated time and pitch spaces are chosen here as an example only; other configurations may combine pitch, time or any other type of data in one data term. The time point is acting as a trigger which triggers successive movements of the pitch point, by sending its subsidiary point \textit{z the message }\textit{next}. \textit{Z} schedules itself by receiving time values from the time shape, in response to movement instructions forwarded to it from the time path. Its time path can be anything from a sequence of duration values to a message object to a system processing input from external sources. The subsidiary point is a very simple object that operates as a pointer marking the position along the time path. In this way, it is possible for many points to traverse the same time path independently at the same time. The time point receives a message from the path, which tells it how to move in the time shape. The time shape refers to the time shape and translates the movement into a specific time value, which is returned to the time point. Based on this time value the time point schedules next movement.

The pitch point operates in a similar way as the time point. Its input is pitch data. The sound \textit{driver} translates this input into instructions in the specific language of the sound generating medium (e.g. csound, MDI etc.)

The subsidiary points and \textit{z} are the crucial connecting elements in this system. In this example, the trigger message \textit{next} is sent by the time point, but it is possible to chain it with other points, into distributing the task of the movement over many modules.
The communication overhead of this approach is big: 15 message sends between modules are required to output a single pitch. This would be computationally expensive if each element were to be modelled as a separate process. But in fact, most of the computations necessary can be implemented as inexpensive function calls or even as operations such as addition, multiplication, etc. So the major issue in the design of the implementation is the trade-off between generality and efficiency.

3 Implementation Example

3.1 Modules of the system

An example of a system based on the music space metaphor, is a "minimal musical language" implemented in MAX with the addition of some external objects. The objective of the implementation was to create a basic language for the algorithmic modeling of music in terms of "musical dynamics", that is in terms of laws governing the movement of abstract elements in space. A bottom-up approach was adopted, aiming at the decomposition of movement into simpler component movements, similar to atomic movements in the operation of a robot. The coordination of these movements is not done by one central program operating on one set of data, but by many programs running concurrently. Two module types are provided:

* Program modules are like virtual microprocessors for executing program code. Each module can contain its own program code, which is editable in a separate window. In addition, the code of other modules is accessible by means of the send message. The program module is a substantially extended version of the MAX cell object.

* Voice modules interpret instructions from other modules into movement in music space and translate this movement to sound and time actions (outputting a note, delaying for a certain amount of time). The voice module is a "hardwared" configuration of a time point and a pitch point with several additional submodules which allow the control of all MIDI parameters such as velocity, control, program etc.

A system may contain any number of modules. The connections between modules can change during program execution. The music spaces used by the system are the tone-net and the duration-net. These represent pitches and durations as points in a multidimensional space. They are infinite lattices of numbers where the ratio formed by any neighboring numbers in each dimension is constant. (The idea for these spaces originates from pitch representations such as found in Vogel 1980, Shepard 1982, Lerdahl 1988 or Krumhansl 1990). The numbers represent pitch frequencies for pitch or time.
units such as milliseconds for duration. The pitch and time relationships of melodic threads moving in these spaces is directly reflected by the relative position of their moving points. As a result, pitch shifting as well as tempo changes can be easily applied to the system as a whole or to any part. Polyphonic structures such as described by Bel (1990, 1992) are implicit in the position of the moving points. So, in contrast to Bel's system, no compilation of polyphonic structures is necessary.

3. 2 Module Communication

Modules communicate with each other via remote connections between output and input ports. Opening a connection between the an output port and an input port means that all messages subsequently sent via the output port will be received by the specified input port. Closing the connection again will stop passing the messages to the input port. An output port may be connected to any number of input ports simultaneously.

The program module has only one input and one output port. On the contrary, the voice module has several specialized input and output ports. The single "main" input and output port communicate with program modules. The other ports share information about the movement of the pitch and duration points directly to the point-moving objects contained inside the voice module.

3. 3 Language Syntax

The basic syntax of the language is very simple: A program consists of a sequence of instructions stored in a program module. Each instruction is a single MAX message stored in one line. To direct messages to subport objects contained in the voice module, the syntax of the MAX message is extended from: `<message name> <arguments>` to: `<message category>` `<message name>` `<arguments>`. That is, symbols may be juxtaposed in a chain, where the leftmost symbol specifies a general message category, and each successive symbol is a more specific message applicable to this category. A message category is a group of messages or message categories which are addressed to different aspects of the same task. For example, the pitch-space category is the group of messages which are addressed to the moving-point object that controls the movement of voice in pitch space. Chained messages are noted here by adding "-" as the end of each leading symbol.

Messages can be grouped into three kinds:

- Messages controlling the execution of a program (e.g. wait, repeat, redo/undo, begin - end).
- Messages to voice modules for controlling sound-parameters, changing the internal state of the module, or triggering processes (e.g. pitch: add <velocity>, note: noteoff, glissando, control: channel-pressure, swell).
- Messages for configuring the interconnections between modules or sending single messages to any module: connect <receive-ads>, disconnect <receive-ads>, etc.

3. 4 The Moving Point Object

The moving-point object is an object responsible for keeping track of the position of a point in 4-dimensional space, and for moving in response to events from other modules. It is therefore the most important part for handling pitch and duration by spatial movement. The voice module contains two identical moving point objects, one dedicated to the position of the voice in pitch space and one to the position during duration. Their input is sent to the corresponding input of the note module. The latter translates the received data into note-messages.

The moving-point object inputs the 4 coordinates of a point in 4-dimensional space, and moves the point by adding to it an internally stored 4-element vector. It furthermore stores the last vector output at slot 0. The operation is defined by six elementary operations:

- *add(+)*: Add vectors a and b:
  
- *subtract(-b)*: Subtract vector b from a.
  
- *store(v)* Store vector v at current position.
  
- *output(v)* Output vector v.
  
- *add(inc,loc,vec)*: Add inc to element loc of vector vec.
  
- *store(val,loc,vec)*: Store value val at position loc of vector vec.

A number of macros combining the above operations are triggered by different messages as for example:

- *list of 4 integers:i0 output(store(i0)).
- *list of 2 integers:loc,inc store(add(inc,loc,vec)).
- *c-loc output(store(inc(loc))).
  
- *store(val,loc,vec).
  
- *set inc loc store(i0).
  
- *b-loc output(vec).
  
- *add inc loc calculate s-vv.
  
- *subtract c-loc calculate s-vv (s is the internally stored vector keeping track of the initial position of the point)

3. 5 Example

This example combines four program modules and two voice modules. The first program (program1) is a combination of main and control program. It sets up the connections of the modules, starts and ends the activity of the system. The second program (program1) is a sequence of 8 pitch-positional delay and related to the initialized origin position of the pitch point. The third program (program2) contains an instruction to change the address of the note module, thereby changing the tempo at each iteration of the phrase. The effect is a repeated gradual acceleration and subsequent deceleration during the phrase. Voice 2 follows the melodic movement of voice 1 but moves at a triple pace. The
program8
[This module configures the system, setting up the connections between modules and initializing the positions of their points. It then counts the output of 100 notes before disconnecting the feedback circle between voice and program and thus stopping the activity of the system]
begin;
send voice1 p set 2 0 0 0
send voice1 i set 2 2 1 0;
set velocity (sound intensity) margins and initial value of voice1
send voice1 v 100 100 10;
select the main output of voice 1 to the program 1, thus always triggering the next movement immediately after the end of each note]
send voice1 connect program1;
[connect the output of voice 1 to the pitch input of voice2, so that voice2 follows the pitch position of voice1]
send voice2 p set 2 0 0 0;
set duration of pitch point in voice1
send voice2 i set 2 2 1 0;
set velocity (sound intensity) margins and initial value of voice2
send voice2 v 100 100 10;
select the main output of voice 2 to the program 2, thus always triggering the next movement immediately after the end of each note]
send voice2 connect program2;
[connect the pitch output of voice1 to the pitch input of voice2, so that voice2 follows the pitch position of voice1]
send voice2 connect voice2;
[connect the output of voice2 to the program 2, thus always triggering the next movement immediately after the end of each note]
send voice3 connect program3;
[connect the output of voice2 to the program 3, thus always triggering the next movement immediately after the end of each note]
send voice3 connect voice3;
[connect the pitch output of voice2 to the pitch input of voice3, so that voice3 follows the pitch position of voice2]
send voice3 connect voice3;
[connect the output of voice3 to the program 3, thus always triggering the next movement immediately after the end of each note]
send program1 bang;
[end of block]
wait 35;
[count 35 notes (≈ 4 repetitions of the 9 note phrase), without producing output]
begin;
send voice1 disconnect program1;
[no more feedback from voice1 to program1 - movement stops]
send voice2 disconnect program2;
[remove feedback from voice2 to program2 - movement stops]
send voice3 disconnect program3;
[remove feedback from voice3 to program3 - movement stops]
end;

program9
[This program causes the pitch point to move starting from its origin point (0 0 0 0) and then to circle around it on each of the neighboring points always at the distance of 1 along each dimension. When reaching the last line in the program, program modules by default restart from the first line.]

p 0 0 0 0;
[p directs the input to the pitch point, plain vector sets and outputs the position of the point]
p 1 0 0 0;
p 0 1 0 0;
p 0 0 1 0;
p 1 0 0 0;
p 0 1 0 0;
p 0 0 1 0;
p 0 0 0 0; 1

program2
repeat 9 0 10;
[Increase the basic pulse of both voices by 10 ms for 9 consecutive times]
repeat 9 0 10;
[Decrease the basic pulse of both voices by 10 ms for 9 consecutive times]

program3
wait 1;
[Wait, 1 ms]

4 Conclusion
This model attempts to formalize musical processes in a way inspired from and possibly suitable for distributed processing, object oriented programming or object based systems and autonomous agents. It also claims at the same time to be an effective way for representing complex musical structures by combining simple elements in a bottom-up manner. The experimental implementation with MAX shows that the basic principle is workable and demonstrates the decomposition of movement into simple constituents. Experiments with the system show that performance is very good because the resulting representation is
computationally economical. On the other hand, encoding by hand is a hard task. The system is not suitable as a general-purpose tool for music composition.

Some aspects of concurrent execution need to be improved, such as the intermediate combination and processing of messages between modules to overcome synchronization problems.

Two directions are open for further research: a) To enrich the representation language with higher level constructs, objects with a richer behavior and interface, and better programming tools (interpreter, code browser, inspector, graphic tools); b) To develop automatic or semi-automatic encoding techniques.

References


